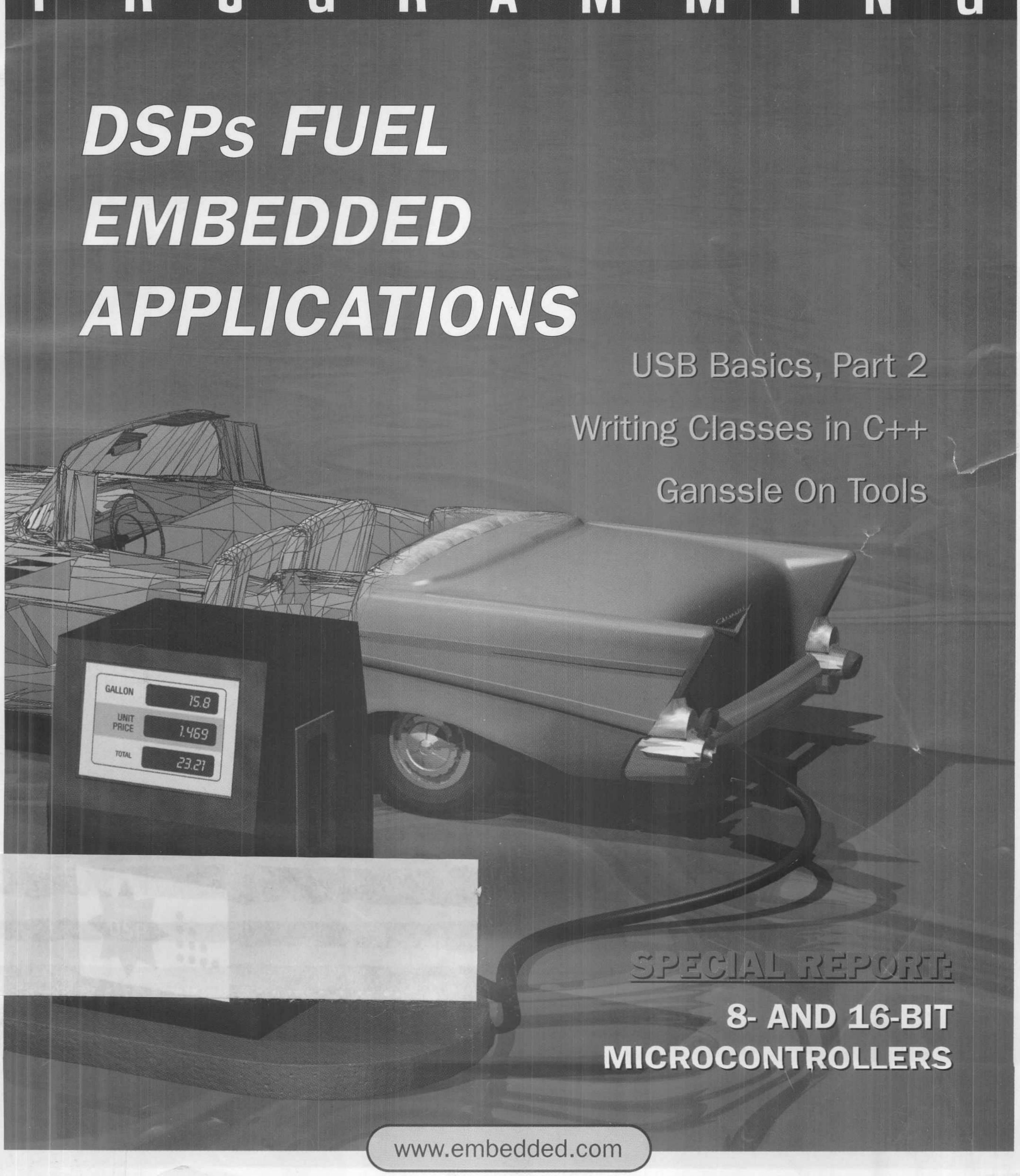
VOLUME 10, NUMBER 7 **U.S. $3.95 CANADA $4.95**
JULY 1997 A MILLER FREEMAN PUBLICATION

Embedded Systems PROGRAMMING



DSPs FUEL EMBEDDED APPLICATIONS

USB Basics, Part 2

Writing Classes in C++

Ganssle On Tools

SPECIAL REPORT:

8- AND 16-BIT MICROCONTROLLERS

www.embedded.com

THE UNIVERSAL CHOICE.

The BP-1400 Universal Device Programmer is easily the most powerful universal programmer available today, supporting over 9,000 devices — an incredible 48% more than the closest competitor. With 240 advanced high-speed pin drivers, the BP-1400 is the only programmer that can test all devices up to 240 pins, including low-voltage devices. The BP-1400's advanced pin drivers produce clean 1ns rise times — the key to accurate programming and trouble-free operation. Over 120 interchangeable socket modules support more devices than any other programmer. Whether you need to program QFP, FPGAs, TSOP, Flash memories, unusual microcontrollers, or old-fashioned PROMs, the BP-1400 won't let you down. With a three-year warranty and free software updates every six weeks, the BP-1400 is the only choice for universal support.

1000 North Post Oak Road • Houston, Texas 77055-7327 • 713-688-4600 • 1-800-225-2102
fax: 713-688-0920 • e-mail: sales@bpmicro.com • web address: http://www.bpmicro.com

BP MICROSYSTEMS

CIRCLE # 1 ON READER SERVICE CARD

No matter where you start, all roads lead to EST.

Boost the power of your debugger interface with visionICE, the emulation system based on a unique, scalable architecture

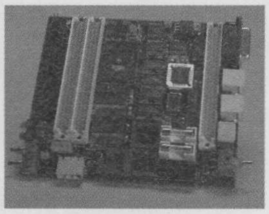
PowerPC

ColdFire

68360

6833X

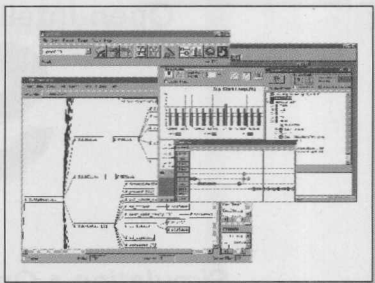
6834X



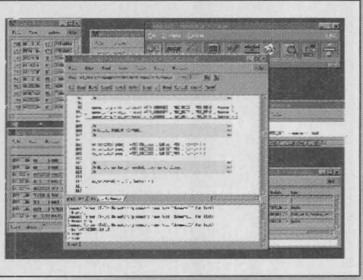
EST's evaluation boards are available for Motorola's 68360, PowerPC 821/860, 6833x, 6834x and ColdFire 520x targets.

It's simple. When it comes to debugging Motorola's PowerPC™, ColdFire®, and 68300 processors, no one gives you more power, more modularity and more software flexibility than EST. No one! Whether you want turnkey evaluation boards, low-cost BDM, network-based BDM, or a scalable ICE complete with real-time trace and a powerful event system, visionICE is your clear choice. But don't take our word for it. Just call and ask any of these highly successful software companies which tools they support to turbo-charge your embedded development environments. You'll see, sooner or later, all roads lead to EST.

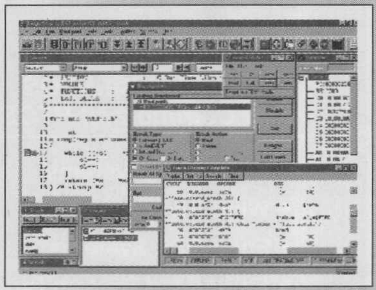
visionICE's scalable design delivers low-cost BDM debugging and real-time emulation.



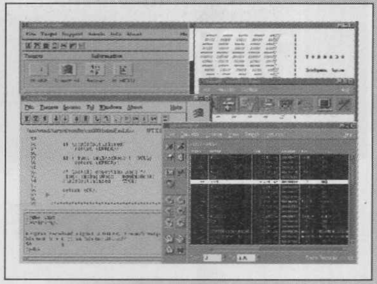
pRISM+™
by Integrated Systems



Spectra®
by Microtec



SingleStep™
by SDS



Tornado™
by Wind River Systems

All trademarks or registered trademarks are property of their respective owners.

800-957-5588

email: sales@estc.com
URL: http://www.estc.com

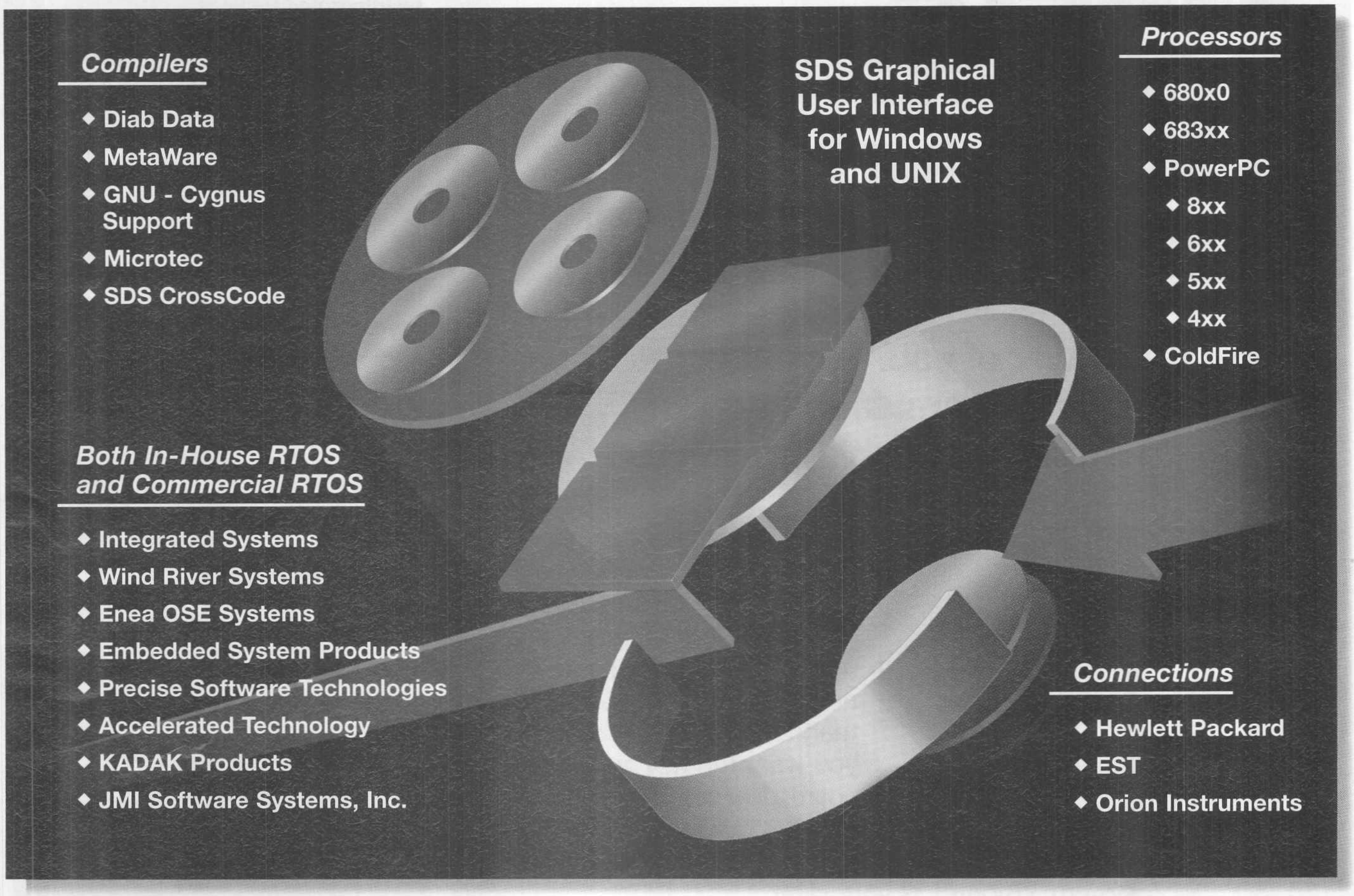
EST

Embedded Support Tools Corp., 120 Royall St., Canton, MA 02021-9725, Tel.: 617-828-5588, Fax: 617-821-2268, European Tel.: 33-130-573200

CIRCLE # 2 ON READER SERVICE CARD

SingleStep™ Debug Suite

Simulation ◆ On-Chip Debug ◆ Emulation ◆ Target Monitors and Servers



Compilers

- ◆ Diab Data
- ◆ MetaWare
- ◆ GNU - Cygnus Support
- ◆ Microtec
- ◆ SDS CrossCode

SDS Graphical User Interface for Windows and UNIX

Processors

- ◆ 680x0
- ◆ 683xx
- ◆ PowerPC
  - ◆ 8xx
  - ◆ 6xx
  - ◆ 5xx
  - ◆ 4xx
- ◆ ColdFire

Both In-House RTOS and Commercial RTOS

- ◆ Integrated Systems
- ◆ Wind River Systems
- ◆ Enea OSE Systems
- ◆ Embedded System Products
- ◆ Precise Software Technologies
- ◆ Accelerated Technology
- ◆ KADAK Products
- ◆ JMI Software Systems, Inc.

Connections

- ◆ Hewlett Packard
- ◆ EST
- ◆ Orion Instruments

Pick the Best, SingleStep Does the Rest.

World-Class ◆ Open Environment ◆ For the Entire Life Cycle

Now you can build your ideal toolset configuration for embedded development. From the components you select. From the vendors you choose. Pick your ideal processor. The fastest compiler. Your favorite real-time kernel. In-house or proprietary. The best debugging tools. Assembled and working together with the latest hardware tools. Integrated with the SingleStep Debug Suite with a common GUI. The SingleStep Debug Toolset is the fastest way to get your next embedded design project up and running—and its open, modular design allows you to keep pace with the latest tools and most advanced processors as they become available. Visit us on the Internet @ www.SDSI.com.

sds
software development systems
The Most Responsive Software Company

815 COMMERCE DRIVE, SUITE 250, OAK BROOK, IL 60521 PHONE 630.368.0400, FAX 630.990.4641.

CIRCLE # 3 ON READER SERVICE CARD

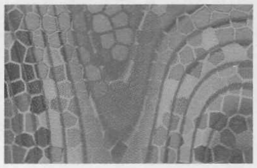
Table of Contents

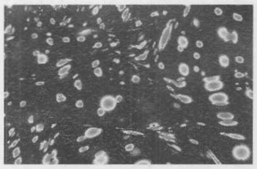
ON THE COVER:

Digital signal processors accelerate embedded applications.

Cover illustration by Rupert Adley.

FEATURES





COLUMNS + DEPARTMENTS

EMBEDDED SYSTEMS PROGRAMMING (ISSN 1040-3272) is published monthly, with an additional issue published in September, by Miller Freeman Inc., 525 Market St., Ste. 500, San Francisco, CA 94105, (415) 905-2200. Please direct advertising and editorial inquiries to this address. SUBSCRIPTION RATE for the United States is $55 for 13 issues. Canadian/Mexican orders must be accompanied by payment in U.S. funds with additional postage of $6 per year. All other foreign subscriptions must be prepaid in U.S. funds with additional postage of $15 per year for surface mail and $40 per year for airmail. POSTMASTER: All subscription orders, inquiries, and address changes should be sent to EMBEDDED SYSTEMS PROGRAMMING, P.O. Box 1278, Skokie, IL 60076-8278. For customer service, telephone toll-free (888) 847-6177. Please allow four to six weeks for change of address to take effect. Periodicals postage is paid at San Francisco, CA and additional mailing offices.

EMBEDDED SYSTEMS PROGRAMMING is available on microfilm/fiche from University Microfilms International, 300 N. Zeeb Rd., Ann Arbor, MI 48106, (313) 761-4700.

Motorola Application-Specified

Digital cordless phones Keyless entry systems Set-top boxes Electronic games Navigation systems Carbon monoxide detectors Computer mice Home security systems Smoke detectors Answering machines Pagers Automotive instrumentation Computer monitors Industrial motor controls Smartcards Home appliances

Over two billion 68HC05 microcontrollers are now in operation around the world. Powering an ever-growing range of consumer, industrial, computing, communications, and automotive products.

Our 68HC05, the world's most popular MCU, is "customer-specified" for your system. In addition to the 150 (and growing) types of 68HC05 MCUs, you can now select one of our fully upward object code compatible 68HC08 microcontrollers.

The 68HC08 Family offers even higher-performance with faster clock speed options and more features for your system design. But the 68HC08 doesn't just provide more instructions and registers, it also runs 68HC05 object code.

Microcontrollers
Industrial motor controls
Smoke detectors Answering machines Pagers
Computer mice Home security systems
Navigation systems Carbon monoxide detectors
Appliances Digital cordless phones
Keyless entry systems Set-top boxes Electronic games
Computer monitors Industrial motor controls Smartcards
Automotive instrumentation

The world's first choice.

And both the 68HC05 and 68HC08 Families offer complete support and a full suite of third-party development tools.

So, if your system needs a "world-class" 8-bit solution, specify the world's first choice: Motorola. Our global manufacturing is already ramped-up to deliver production volumes anytime, anywhere.

For more information on either the 68HC05 or 68HC08 Families, call **1-800-765-7795** ext. 882 or request by FAX at **1-800-765-9753**.

You may also visit our Web site at **http://sps.motorola.com/csic**. When it comes to 8-bit solutions, there's a world of difference with Motorola MCUs.

MOTOROLA
CSIC Microcontroller Divisions

What you never thought possible.™

Visit us on the World Wide Web

http://www.keil.com/

KEIL SOFTWARE

Embedded Software Development Tools

For the Siemens 166, C167, C165, and C163 microcontroller family and the NEW! C161

C166 Version 3.0 — The High-Performance C Compiler for the Siemens 166 Family

Keil Development Tools unlock the features and performance of the Siemens 166 and C167 and make you an instant 16-bit embedded expert. The µVision Windows-based IDE encapsulates your project with complete compiler and linker controls and helps you create complex programs in record time! The dScope symbolic, source-level debugger lets you simulate and debug your target system including external hardware.

The ANSI-standard C166 compiler is designed specifically for the 166 and C167 families—language extensions give you access to all CPU resources including the PEC, interrupts, SFRs, and DPPs.

C166 is the most efficient, flexible development tool set available today. Support for all derivatives and compatibility with the major emulator vendors makes C166 the best choice for your 166 and C167 projects!

PK166/PK161 Highlights

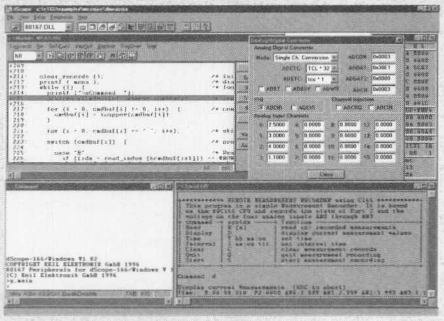
- ✓ ANSI-compliant C Compiler with C Libraries
- ✓ Includes Memory Allocation Routines
- ✓ Floating-Point Libraries Included
- ✓ Support for Structures and Unions
- ✓ Interrupt Functions may be written in C
- ✓ Reentrant Functions
- ✓ Parameters Passed in Registers
- ✓ C Support for all Special Function Registers
- ✓ Supports the Entire 16M Address Space
- ✓ Inline Assembly Code Support
- ✓ Includes Macro Assembler
- ✓ Includes Single-Chip Real-Time Operating System
- ✓ Windows-based IDE and Debugger
- ✓ Free Updates via the World Wide Web
- ✓ Free Update Notification via E-mail
- ✓ Free One-Year Technical Support

dScope Source-Level, Symbolic Debugger and Target Monitor

The dScope source-level, symbolic simulator/debugger helps you test and debug your 166 and C167 application programs. dScope provides:

- Execution, conditional, and memory access breakpoints,
- Watchpoints for all variable types,
- Mixed source/assembly display,
- Software performance analysis,
- Code coverage analysis,
- User and signal functions,
- and On-chip peripheral support.

Debugging your target hardware is easy when you use the MON166 monitor and dScope debugger. MON166 is a full-featured, royalty-free target monitor designed for the 166 and C167 families. It can be configured for a wide variety of systems—even those with bootloader capabilities. Using dScope and MON166, you can easily view program source code, watch special variables, and examine target memory! And, MON166 comes preconfigured for a variety of third-party evaluation boards.



```
dScope-166/Windows V1.02
COPYRIGHT KEIL ELEKTRONIK GmbH 1996
80167 Peripherals for dScope-166/Windows V
(C) Keil Elektronik GmbH 1996
```

dScope provides you with a powerful debugging platform that completely simulates **all** 166 and C167 derivatives including the C165, C163, and C161 devices. Using CPU driver DLLs, dScope gives you dialog box access to all on-chip SFRs. You can easily view and modify SFR contents for A/D converters, timer/counters, ports, and serial ports.

RTX166 — Real-Time Operating System

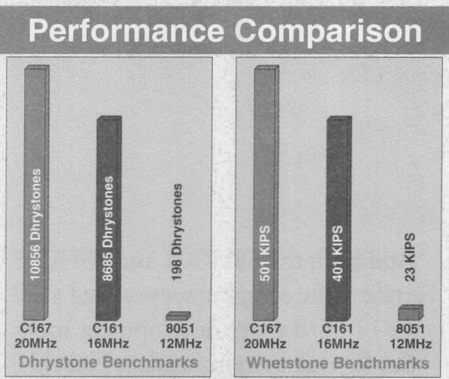
RTX166 is a multitasking real-time operating system that supports the entire Siemens 166 and C167 family. RTX166 makes designing complex, time-critical software projects easy by providing sophisticated management for multiple tasks running on a single CPU.

RTX166 Tiny supports single-chip applications where code and memory space must remain at a minimum. RTX166 Tiny lets you create and delete tasks and send and receive signals.

RTX166 Full supports applications where robust features are required. In addition to the features found in the tiny RTX, the RTX166 Full version manages interrupts, resources (via semaphores), and memory pools. Mailbox, system clock, and task management routines are also included.

RTX166 Full provides support for the C167CR CAN interface. This lets you get started with the CAN interface as quickly as possible.

Performance Comparison



Benchmark measurements were made using the performance analyzer in dScope.
C167 and C161 programs were compiled for 16-bit, non-multiplexed bus mode with 0 wait-states.

Call 1-800-521-4957 for a FREE Evaluation CD-ROM of all our tools!

United States:
Keil Software, Inc.
16990 Dallas Parkway, #120
Dallas, Texas 75248-1903
USA

Phone	972-735-8052
Fax	972-735-8055
BBS	972-713-9883
E-mail	sales.us@keil.com support.us@keil.com

Europe:
Keil Elektronik GmbH
Bretonischer Ring 15
85630 Grasbrunn
Germany

Phone	++49 89 / 456040-0
Fax	++49 89 / 468162
BBS	++49 89 / 4606286
E-mail	sales.intl@keil.com support.intl@keil.com

US Distributors: **Ceibo:** 314-830-4084, **CMX:** 508-872-7675, **Emulation Technology:** 408-982-0660, **HiTools (Hitex):** 800-454-4839, **Metalink:** 602-926-0797, **Micom Systems:** 214-245-2533, **Microware Technology:** 619-693-4280, **Nohau:** 408-866-1820, **Peachtree Technology:** 770-888-4002, **Signum Systems:** 805-371-4608.

International Distributors: **AUS:** Electro Optics 02/9654 1873, **A:** Rekirsch 01/259 72 70 0, **B:** Bytecom 010/22 34 55, **Brazil:** Anacom 11/453 5588, **Canada:** Ximetrix 905-602-8550, **Czech:** ComAp 2/6833 858, EDI 2/02 2683, **CH:** Thau 01/745 18 18, **DK:** Nohau 043/44 60 10, **E:** CAPEL 93/291 76 33/34, **F:** Antycip 1/3961 1414, **Hong Kong:** Oritental 852/2402 3200, **I:** Microtask 02/4982051, Nord 02/66092.1, **India:** Embedded Systems Solutions 80/3323029, **IRL:** Ashling 061/33 44 66, **Israel:** ITEC 03/6 49 1202, **Japan:** NPS 03/3226 8110, **Korea:** Hankook 02/783 0022, **Malaysia:** Flash 603/7168729, **NL:** Tritec 078/681 6133, **NZ:** Electro Optics 0800/44 2148, **PL:** WG 02/621 77 04, **Romania:** ASTI 3125907, **S:** Nohau 040/59 22 00, **Singapore:** Flash 65/749 6168, Testech 65/7492162, **SLO:** ASYST 061/445526, **S. Africa:** EBE 012/803 7680/93, Kiberlab 012/660 2752, **Taiwan:** Deemax 3/5232548, **Turkey:** EMPA 1/599 30 50, **UK:** Hitex 01203/69 20 66, Nohau 01962/73 31 40.

CIRCLE # 4 ON READER SERVICE CARD

Embedded Systems

MANAGING EDITOR
Mariska van Aalst

TECHNICAL EDITOR
Nicholas Cravotta

COPY/PRODUCTION EDITOR
Michael Shapiro

CONTRIBUTING EDITORS
Jack W. Crenshaw
Jack G. Ganssle
Larry Mittag
Don Morgan
P.J. Plauger
Tyler Sperry

PUBLISHING DIRECTOR
Donna J. Esposito

PUBLISHER
Mike Flynn, (415) 278-5251

NATIONAL SALES MANAGER
Eric Berg, (415) 278-5220

CA REGIONAL SALES MANAGER
Thomas Vilms, (415) 278-5223

WESTERN SALES ASSISTANT
Robin Lander, (415) 278-5274

EASTERN REGIONAL SALES MANAGER
Damon Graff, (617) 235-8258

EASTERN SALES ASSISTANT
Ryan Sorley, (617) 235-8255

PRODUCTION COORDINATOR
James Whitehead

GRAPHIC DESIGNER, ELECTRONICS
Katie Symons

GROUP CIRCULATION MANAGER
Sherri Gronli

ASSISTANT CIRCULATION MANAGER
Jennifer Armstrong

SUBSCRIPTION CUSTOMER SERVICE
Toll Free: (888) 847-6177
Non-U.S. subscribers: (847)647-8602
embedded@halldata.com

REPRINTS
Andrea Ramiza, (714) 376-6273

Visit the *Embedded Systems Programming* Web Site
www.embedded.com

Miller Freeman
A United News & Media publication

CHAIRMAN/CEO
Marshall W. Freeman

EXECUTIVE VICE PRESIDENT/COO
Donald A. Pazour

SENIOR VICE PRESIDENT/CFO
Warren "Andy" Ambrose

SENIOR VICE PRESIDENTS
H. Ted Bahr
David Nussbaum
Wini D. Ragus
Darrell Denny
Galen A. Poss
Regina Starr Ridley

VICE PRESIDENT/PRODUCTION
Andrew A. Mickus

VICE PRESIDENT/CIRCULATION
Jerry M. Okabe

SENIOR VICE PRESIDENT ELECTRONICS/DESIGN DIVISION
H. Ted Bahr

BPA INTERNATIONAL

Anomalies Unbound

An old joke asks, "Who invented cottage cheese? And how did they know when they were through inventing it?" Software is like cottage cheese in the sense that it's hard to tell when it's done. The same is true of almost any sort of intellectual content. Fiction writer Flannery O'Connor was never satisfied with anything she wrote and was only published when her editor pried her manuscripts from her hands. Mike Nichols spent a year editing *Catch-22*, morphing the film from an epic with a cast of thousands into a metaphorical piece whose cast of thousands wound up on the cutting room floor. John Fogerty, of CCR fame, has been described as a perfectionist who drives the production cost of his albums up because of his meticulousness.

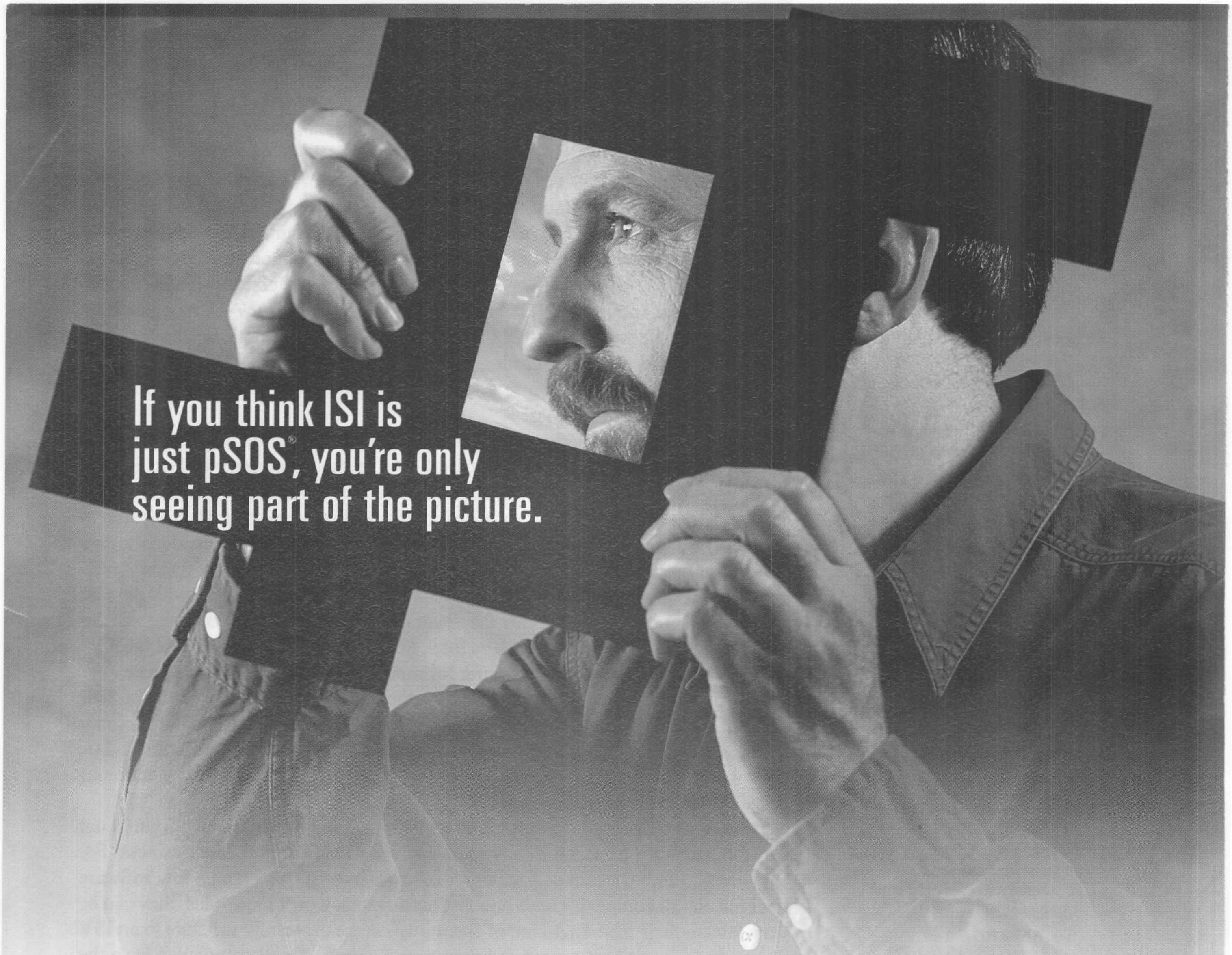
It's hard to tell when software is done, because it's hard to do. Eighty percent of a system's complexity is in software, and 80% of the development effort is software. With the increasing importance of software as the source of product differentiation, it seems curious that software companies are often so small compared to the hardware companies they support. While there are obvious exceptions to this rule of thumb, it is clearly evident among purveyors of embedded software development tools. Compare the size of the companies that build the tools with the size of the semiconductor companies they support. Tool vendors are often so minuscule, in contrast to the giants, such as Intel, Motorola, and TI; they are like the pilot fish that hover around sharks, waiting for morsels of food to come their way.

Software companies have often relied on the success of a single product for their survival. The classic example is Micropro International, whose WordStar program dominated the word processing market, and along with Visicalc and Lotus 1-2-3, created the PC market. Wherever you look, software is what translates electronics into a successful application.

Despite the value of software, most semiconductor companies have figured out that the hardware business is way more lucrative than the software business. For some reason they'd rather sell hundreds of thousands of chips than a few software licenses. Small companies have one distinct advantage over large ones: they can be very fast on their feet. That kind of flexibility can be especially valuable to an ISV during times such as these when innovation abounds. Small companies have the freedom to be innovative and responsive. Not only can they bring new tools to market quickly, but they can bring new types of tools as well to accommodate more complex design problems. As we're developing our 1997 *Buyer's Guide* issue, we're finding it a challenge to invent enough categories to accommodate all of the tools we want to include.

Speaking of challenges, we're looking for a new technical editor. Nicholas Cravotta is being pried away from *Embedded Systems Programming* by the publisher of a startup Miller Freeman publication. Finding someone conversant with embedded system development who can write about that subject in a logical, coherent way is not easy. If you live in the San Francisco Bay Area and have an inclination to explore the wacky world of publishing, drop me a line. You can help us invent each issue—and tell us when we're through inventing it.

lvereen@mfi.com


If you think ISI is just pSOS®, you're only seeing part of the picture.

We make your job easier by delivering much more than just pSOSystem™— the industry's leading RTOS. The fact is we provide the greatest breadth and depth of technology, tools and services in the embedded systems industry.

And as new technologies emerge, like Internet and Java,™ you can be sure we're one step ahead. Because we're always looking at the big picture.

Our range of solutions includes:

RTOS
pSOSystem™: Reliable, scalable RTOS, proven across the industry's largest installed base.

Tools
pRISM+™: Powerful, integrated graphical software development environment.

Network and Application Components
Epilogue™: Industry's leading platform-independent protocol software.
pSOSystem Components: Industry-specific application modules.

Engineering Services
Doctor Design: Foremost providers of innovative design solutions.

integrated systems
www.isi.com
800.543.7767

Integrated Systems, Inc. 201 Moffett Park Drive, Sunnyvale, CA 94089

pSOS is a registered trademark and pSOSystem, pRISM+ and Epilogue are trademarks of Integrated Systems, Inc. Java is a trademark of Sun Microsystems, Inc.

CIRCLE # 5 ON READER SERVICE CARD

by Tyler Sperry

Oh, Those Pesky Details!

After years of loyal patronage, the ever-degrading performance of my ISP (Netcom) finally forced my hand. For months, I'd been playing the telecommunications equivalent of Russian Roulette, pulling the redial trigger in a desperate effort to connect to a server that could send more than half a line of text without pausing for a few minutes of rest. At one point I became convinced that the provider was attempting to emulate the throughput of one of those old-fashioned Teletypes. And a pretty good emulation it was! While I missed the sound of a mechanical printhead going chunkity-chunkity-chunkity across the page, Netcom's attempts to emulate 55cps throughput on a T-1 link were every bit as aggravating as the original.

The most immediate result of all this was my shifting to a new e-mail address. I have little doubt that the change will be quickly verified by the loyal cadre of readers who selflessly e-mail me with the endless litany of my errors in fact and judgement. Thank you in advance. Having passed some baksheesh to the Lords of the InterNIC and hence established my very own domain name, I hold to the naive hope that future server problems will require only a change of hardware instead of a change of address with its attendant hassles. It sounds good in principle, but then so do most portability schemes. The devil, as they say, is in the e-mail. Or was that the details? I forget which.

Another change: your humble, disobedient servant has recently taken to working at a Web site. The business has nothing to do with embedded systems, so there's little chance of a conflict of interest. Of course, given the rapid pace of most venture-capital-funded startups, by the time you read this I could either be at work fostering an online developer community, or I could exist as a mere footnote in a sysadmin's cleanup log.

For a second, it looked as if Gates and McNealy would settle their differences with a good old-fashioned fist fight.

I wouldn't have bothered mentioning that Web item except for its fallout. Spending a good part of each day connected to the net has changed some of my perceptions on the Web's utility. As befits the somewhat conservative nature of the embedded software world, I'm still searching for some really good embedded links. When I find some, I'll be happy to pass them along here. Indeed, by the time you read this, I hope to have this column on the *Embedded Systems Programming* Web site (www.embedded.com/current.htm). The viability of Web-based columns may be questionable, but the Web does offer the advantage that you can just point and click to follow up on this column's HTML links instead of making typing errors while trying to decipher an address obscured by a coffee stain.

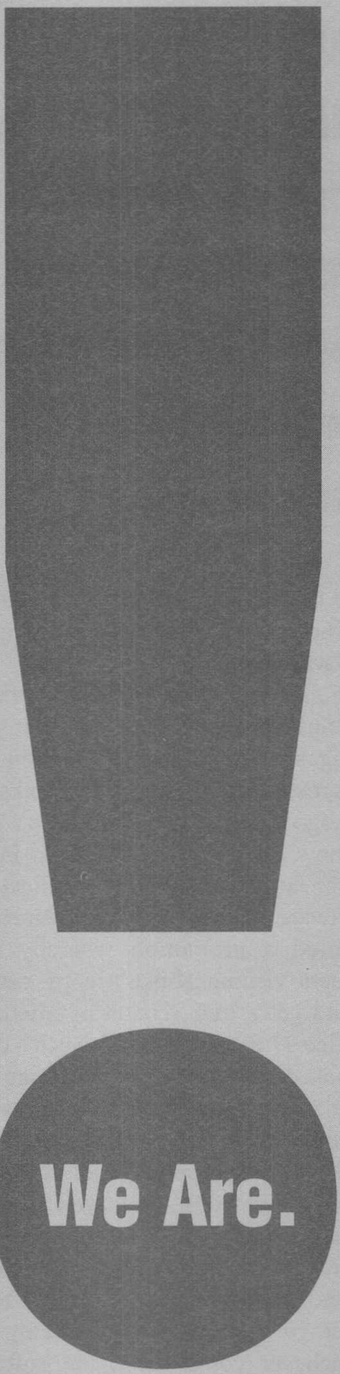
MARATHON MEN

This spring saw a virtual rumble in San Francisco. For reasons known only to Loki, the ancient god of mischief and group scheduling, the Spring edition of Software Development Conference was held in one section of the Moscone Center at the same time Sun Microsystems was holding Java One in another. For a brief second, it looked as if Bill Gates and Scott McNealy would finally settle their differences in the manner of grade school children everywhere, with a good, old-fashioned fist fight. But alas, instead it turned into the usual "Am not!" "Are too!" exchange, with members of the press cheerfully carrying the indirect responses back and forth.

To be sure, things started out with what looked like a poke in the eye for Gates and company. Scott McNealy of Sun made a big deal of a staged "demonstration" of how a wicked witch of the Web might subvert your system using an ActiveX program to read confidential information from your hard drive. There was an element of truth in the demo, in that a clever (and malicious) software developer could make some reasonable assumptions about file locations and do Bad Things with the information.

The problem with Sun's demo is that virtually any software that does useful things will require access to a client's file system, and therefore is potentially just as dangerous, including applets using the next generation of Java. Sure, you're perfectly safe using an applet in Java's "sandbox," but then, you're also perfectly frustrated from getting any work done. Worse, security problems such as the ones McNealy staged are just as likely to pop up in Navigator plug-ins or in the next generation of Java applets. Since McNealy was undoubtedly aware of this, his exposé of the "holes" in Microsoft's security model is revealed as one part fact, two parts con job.

In keeping with this month's Web theme, I invite you to check out Microsoft's response at www.microsoft.com/security/actxclar.


We Are.

htm. Their white paper has links to the relevant "trust" security models offered by Microsoft, Netscape, and Sun. You can compare the models yourself. While such issues might seem unconnected to today's embedded systems, the recurring theme of security holes is worth tracking if your project is intended to connect to an unprotected network.

NON-NEWS, AS IT HAPPENS!

Without a doubt, the biggest non-story of Java One was the announcement of an impending spec for Embedded Java. For those members of the technical press who had failed to notice the huge market demand for Internet-linked hotel door locks, Sun was offering a second chance to climb aboard the Java bean wagon. Indeed, months before the Embedded Java spec would be revealed to the world, Sun was trumpeting the news that both Schlumberger and Bull had licensed Java, presumably for the next generation of smart cards. McNealy noted at the conference that "Gemplus sells one million smart cards per day, including weekends."

So what's wrong with this picture? Well, to start with, most of the existing smart cards are running low-cost processors like the 68HC05. Expecting those applications to suddenly upgrade to software requiring a couple of megabytes seems, well, crazy. A Java license is a small expense for any company that wants to position itself as being on the leading edge of software technology, but actually deploying the technology in products is another thing. Put me down as a skeptic on this one. Especially because every example application the embedded Java pundits have offered is already being addressed with much lower-cost microcontroller systems.

NO FREE RIDES

While I do understand some of the inherent difficulties of internet security, I can't let Microsoft off the security hook just because Sun also has some problems. Indeed, the past non-embedded track record of these companies is one of the reasons that their overtures toward the embedded world give me the willies.

What I found disturbing was not that Microsoft's Office 97 CD was breakable, but the reported ease of breaking it.

Following up on Bill Gates remark at Software Development that languages are a tiny fraction of Microsoft's business (Gee thanks for the ego strokes, Bill!), let's take a look at how the company has fared in the more lucrative area of applications software. This spring, Microsoft offered a promotion whereby customers visiting Kinko's copy centers could get a trial version of Microsoft Office 97 for a measly $5. Such a deal! Plunk down some pocket change and you get a fully-functional release of Word, Excel, Power Point, and other disk padding applications. It sounds like a righteous deal, you say, so where's the catch?

The intended catch is the expiration date. The package is designed so that after 90 days of use or on July 1st, whichever comes first, the software will stop working. Microsoft's reasoning is that users will be so enamored of the software (or so lazy) that registering and buying a fully-licensed copy will be a compelling deal.

The *real* catch is somewhat different. Shortly after the CDs had started appearing in Kinko's everywhere, an enterprising reporter noticed that there was a "crack" for the software. Indeed, fire up just about any Web search engine and you will discover an entire subculture of renegade programmers who are obsessed with breaking the copy protection for newly-released software. It didn't take very long for the Office 97 suite to appear on that list.

What I found disturbing was not that Microsoft's Office 97 CD was breakable, but the reported ease of breaking it. Here is Microsoft's premier applications software, representing a large development investment, and their protection consists of a simple date check buried in a DLL. The protection scheme was so primitive in fact, the programmer who broke it reported some disappointment that the whole process of finding the routine and creating a patch took less than an hour. He had expected better security from Microsoft.

When you combine this story with the running soap opera of holes discovered in Internet Explorer, a certain pattern begins to emerge. As a resident of the politically-correct San Francisco Bay Area, I must deny myself the all-too-appropriate remark about kicking the "differently-abled" coders in Redmond. Instead, I will simply leave the topic with a challenge to you, the readers: pass along your examples, if you have any, of security areas that Microsoft hasn't bungled and I will consider them for a future column.

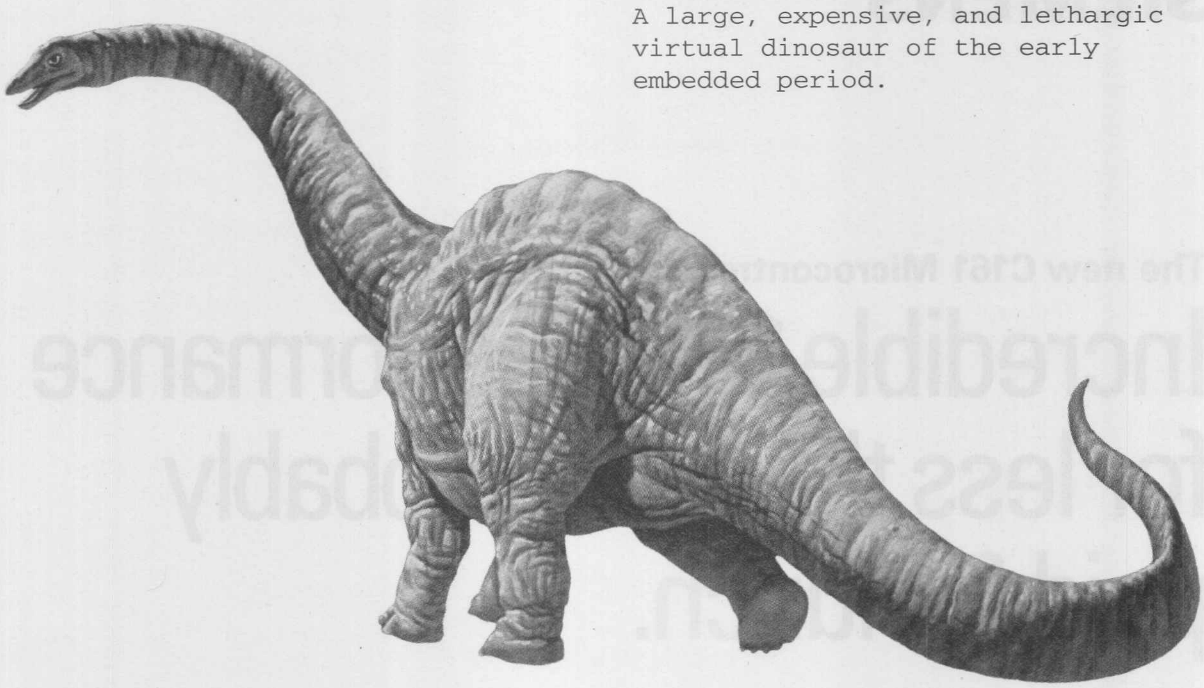
I will only add the qualification that Microsoft employees are exempt from the contest. My fear is that they are likely to have notions of well-implemented security that are not shared by many readers. We're talking, after all, about a company that can refer to Windows95 as a multi-user OS because you're allowed to quit a session and then login under a different name.

A FEW GOOD LEADS

Of course, keeping track of details is a full-time job when it comes to the x86 architecture. I had thought that by now, with the 386 having safely "matured" into the realm of embedded products, that we'd begin to see some signs of stability in the marketplace. No such luck.

As I write this, yet another company is threatening to announce their entry in the x86 market. Lured by the vision

RTOS•sau•rus (ar-tos-sor-es) n.
A large, expensive, and lethargic
virtual dinosaur of the early
embedded period.



Don't let extinct kernels take your product development into the Ice Age...

Express Logic can lead you to a new age of embedded systems development with ThreadX™—a real-time kernel designed for real-time applications like yours.

Size. ThreadX has a microscopic footprint—typically requiring under 3Kbytes of instruction-area memory.

Speed. Because of its advanced picokernel™ architecture, ThreadX achieves the fastest possible performance.

Intelligence. ThreadX is top of the order. It features preemption-threshold™—a completely new technique to reduce context switching. For more information, ask us for a free whitepaper!

Easy to use. ThreadX is a snap to use! Check out our API to see for yourself. In addition, you get full C source code should you need to customize it.

Processor support. You name it. ThreadX supports most popular processors, and we're adding new ones all the time.

Business friendly. Absolutely. ThreadX kernels are offered at reasonable prices and—most importantly—without any run-time royalties!

To find out how you can speed up your product development with ThreadX, call Express Logic today.

eL
EXPRESS LOGIC

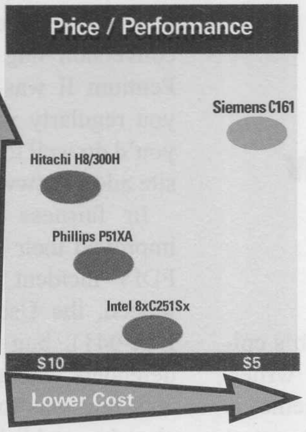
toll free **1-888-THREADX**
1-619-674-6684
info@expresslogic.com
www.expresslogic.com

CIRCLE # 7 ON READER SERVICE CARD

SIEMENS

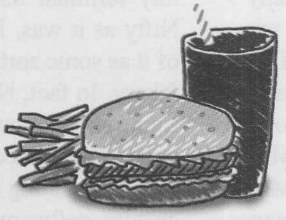
The new C161 Microcontrollers.

Incredible 16-bit performance for less than you probably paid for lunch.



Satisfy your appetite for true 16-bit — without compromise.

If you find upgraded 8-bit products pretty hard to swallow, Siemens has a treat for you – the new C161 microcontrollers in stock today. All are based on the C166 core with RISC-like architecture, with an instruction set and optimizing compilers designed for easy C programming.

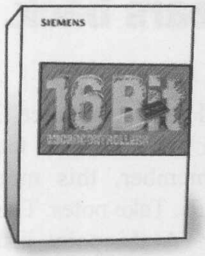


A small price for mouth-watering performance.

At just $4.79 each at 1,000 pieces, the C161V delivers true 16-bit architecture for the cost of 8-bit products. So the C161 microcontrollers give you the performance level you want without taking a huge bite out of your budget.

A full menu of development tools available now.

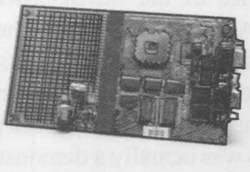
Our worldwide group of established partners offer a complete range of stable tools to take your design through to finished product in short order.



A tantalizing $161 for our stuffed-full Evaluation Kit.

The complete C161 evaluation kit.

Includes Evaluation board, Tasking, Keil and HighTec C-compilers, Assemblers and Debuggers, along with comprehensive documentation.



C161 Evaluation Board.

With 64 kByte RAM, 256 kByte FLASH operating at full speed, it's the perfect starter for your most innovative ideas.

Manuals and CD-ROM

Includes everything you need: Development Tools, User's Manuals, Data Sheets, Application Notes, Web pages and Links.

For more information and to order your C161 Evaluation Kit (SABC161EVAL), call our distribution partner, Marshall Industries at 1-800-877-9839 ext. 3109.

For sales service 24-hours a day, seven days a week, call 1-800-833-9910 or visit www.marshall.com.

Marshall

IT'S ABOUT TIME

©1997 Siemens Components, Inc. www.sci.siemens.com/mc.html

CIRCLE # 8 ON READER SERVICE CARD

of all those network computers to be sold, the company is scheduled to announce their family at the PC Tech Forum, a week away as I write this. As you might imagine, Intel wasn't particularly shy about suggesting that any new x86 products would be rigorously examined to make sure they don't infringe on Intel's intellectual property. This statement took on a richly ironic note when both Digital Equipment and Cyrix announced they were suing Intel for infringing on their patents. If tradition holds, we can expect another long, expensive period of legal wrangling where the lawyers argue over who really invented the phase-locked loop or some similar matter. It's hard enough to keep track of the technical details without adding all those legal nuances to the mix.

And speaking of technical details, who better to help you understand the performance of a new processor than the company that's selling it, right? I noticed at the Winter Consumer Electronics Show that Intel was proudly showing off systems with the new MMX technology. A 200MHz Pentium Pro with MMX zips right along and makes for a great game demonstration. Still, I couldn't help but wonder how much of that sizzling performance was due to the clock speed rather than MMX. By some curious turn of events, Intel's small room of demo machines didn't include a side-by-side comparison of MMX-enabled and MMX-bereft systems running at the same clock speed, and I was forced to leave with that question unanswered.

I wasn't the only one with such questions. Bob Cringely also had some questions. Which Cringely, you ask? Good question. There is a person writing *Infoworld*'s techy gossip column under that name, whom I shall refer to as "Bob #4," and then there is the person who used to write that same *Infoworld* column, also under the name of Robert X. Cringely. I shall refer to the latter as "Bob #3" or "PBS Bob." just to keep things confusing. Suffice to say, Bob #4's column in *Infoworld* offers a larger selection of useful technical gossip, but then, PBS Bob's column proves he is the better writer. (Remember, this month's theme is details. Take notes. There may be a test later.) At this point, just to make things more confusing I am tempted to bring up "Lame Bob," the over-friendly offspring of Microsoft's marketing and user-interface groups. Alas, I promised to stop picking on Microsoft earlier. So I'm not going to mention him. Really.

In fairness to Intel, they have improved their act since the infamous FDIV incident of a few years back.

Back to those performance details. PBS Bob noticed that there was a discrepancy between the glowing performance of the MMX-powered version of Adobe Photoshop in the company's performance benchmarks and what users might actually see in real life. It turns out the 400 to 800% performance gap was actually a demonstration of the worst-case situations for non-MMX machines when compared to MMX performance. The typical performance boost provided by MMX was actually in the range of 8 to 12% rather than the triple digits. (See www.pbs.org/cringely/archive/apr2497_main.html for details.) This would just be another "danger of interpreting benchmarks" story if it weren't for the punchline: Intel's engineers provided the cooked benchmark code to Adobe. Can you say "faux pas?" I knew you could!

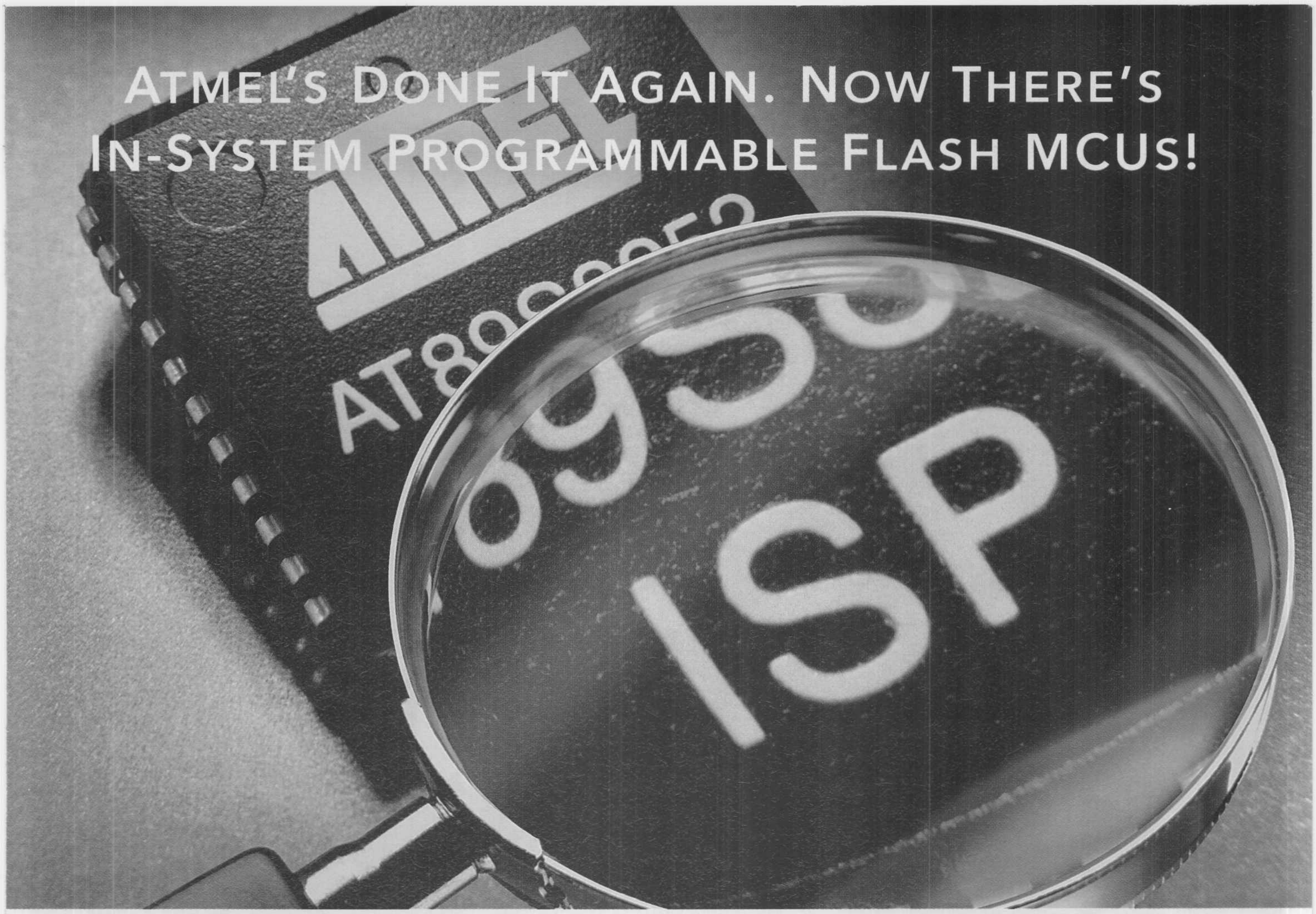
OH, WHAT A WICKED WEB!

Although the Bobsey twins offer the occasional technical tidbits, for hardcore Intel watchers, the site to check regularly is operated by Robert Collins. Collins has made a reputation for regularly posting information that makes Intel managers squirm. So much so, in fact, that his home page visitor count keeps track of ordinary visitors and visitors using Intel accounts. His most recent exploit was breaking the news of the "Dan-0411" floating-point conversion bug just days before the Pentium II was officially released. If you regularly work in the x86 realm, you'd do well to put a bookmark on his site address (www.x86.org).

In fairness to Intel, they have improved their act since the infamous FDIV incident of a few years back. Indeed, the Usenet discussion of the Dan-0411 bug included some Intel defenders who argued that the alleged new bug was in fact a new report of already established bugs #20 or #46 from Intel's erratum sheet. This is progress. I think.

Finally, for those who wonder about my repetitive mentioning of the x86 architecture, let me assure you we still haven't begun to see the full range of embedded applications this architecture can support. For example, did you see this spring's movie, *The Saint*? In that movie, Val Kilmer had a magic cellphone that would flip open to reveal a tiny terminal and QWERTY keypad. Nifty as it was, I was inclined to think of it as some sort of James Bond movie fakery. In fact, Nokia is already selling such phones outside the USA. The units are powered by, ahem, an embedded 386 running GEOS and a variety of custom software. While it's true that the $12,000 price tag puts such phones out of the impulse buying category, it's important to remember the big picture. Just think how much cheaper they'd be with a little extra memory and a Java virtual machine to run things. You might laugh, but I'm assured by the Sun worshipers that such products are in the works. The only delays, I'm told, will be to work out a few technical details. And so it goes. ESP

Tyler Sperry lives and breathes details. Contact him electronically at tyler@nlper.org.

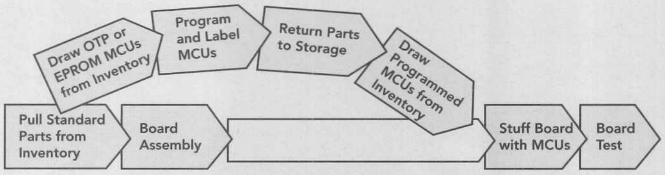

ATMEL'S DONE IT AGAIN. NOW THERE'S
IN-SYSTEM PROGRAMMABLE FLASH MCUs!
AT89S
ISP

GET TO MARKET FASTER! ATMEL'S 8-BIT FLASH MCUs CUT MANUFACTURING TIME BY ELIMINATING THE UV-ERASE CYCLE AND STREAMLINING YOUR PRODUCTION LINE. Atmel's AT89S family of MCUs are programmed via the industry-standard Serial Peripheral Interface, so whether it's a field upgrade or a last minute change in manufacturing, your product is a success. Plus, they're footprint and code-compatible with 80C52 products. In addition, the AT89S8252 has 2K bytes of on-chip EEPROM data memory for better integration and improved system security.

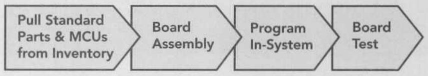
Part	Flash	On-chip EEPROM	ISP
AT89S8252	8K	2K	Yes
AT89S53	12K	No	Yes

Someday everyone will build microcontrollers this way.

Today, get them from Atmel. Call **1-800-365-3375** for more information from the Flash MCU company.


Pull Standard Parts from Inventory
Board Assembly
Draw OTP or EPROM MCUs from Inventory
Program and Label MCUs
Return Parts to Storage
Draw Programmed MCUs from Inventory
Stuff Board with MCUs
Board Test

The old, time and parts consuming method.



Streamlined flow made possible by Atmel's new Flash MCU devices.

E-MAIL: literature@atmel.com
FAX-ON-DEMAND:
(800) 292-8635 (North America)
(408) 441-0732 (International)
WEB SITE: www.atmel.com
CORPORATE HEADQUARTERS:
2325 Orchard Parkway, San Jose, CA 95131
TEL: (408) 441-0311 FAX: (408) 487-2600

Hitachi ICs bring your biggest ideas down to size.

IC solutions for the hottest applications from 2-way wireless paging to Windows® CE Handheld PCs.

These days, the biggest ideas in consumer electronics are all about the same size: Handheld. They're now the "Wow" of Wall Street; the "Egad" of editors. And they're the electronics industry offering a hand to millions who have not yet "gone digital." For OEMs of successful Personal Access products, their ever-increasing integration within the size, power and cost constraints of handheld systems is good reason to shake hands with Hitachi.

How to get bigger, better, smarter, smaller. Thanks to Hitachi's ability to combine its best-selling line of MPUs, MCUs and advanced memory devices, and deliver these as integrated solutions, we have become the leading IC supplier for handheld systems. In fact, Hitachi's SuperH RISC Engine is the processor of choice for the overwhelming majority of the new Windows CE Handheld PCs.

Hitachi helps you hit the small time. To learn how you can get small fast, phone 1-800-446-8341, ext. 800. Or visit our web site at **www.hitachi.com.**

At Hitachi, we understand that the trick is not to think big; the trick is to think big, then to think really, really small!

HITACHI

#1 in RISC Shipments

by Jack W. Crenshaw

Where's He Been?

This month's column is not going to be one of my usual ones. I won't be talking about integer arithmetic or other math algorithms, but rather about what happens when Murphy's law takes over with a vengeance; when the events with seven-sigma probability take charge.

If you've been a faithful reader of this column, you surely have noticed that it's been absent for a couple of months. I'm sorry about that, but be assured, I wasn't vacationing on the Riviera or any such thing. Believe me, it wasn't nearly that much fun. What happened was a series of computer disasters, all at once. These disasters included a sudden and total hard disk crash on my home machine—with no backup. I also endured an equally total crash on my development system at work. Likewise, a hardware failure on the third, target machine, plus lots of other interesting twists, like a column sent by e-mail, but received in unreadable form. Ever have one of those really bad days?

A head crash is hardly the kind of thing to write a column about, but I think this bizarre combination of events may definitely be worth your attention, if only as an object lesson for what can happen when one gets too complacent, and how to plan for the worst case, seven-sigma situations. In the process of recovering from the disasters, I've also learned some neat things, which I'll pass along in the hopes that you too will find them useful. We'll get back to integer multiplication next month.

What happened was a series of computer disasters, all at once—ever have one of those really bad days?

THE SCENE IS SET

Before I tell you what happened, I must set the scene by explaining the environment I operate in. Most working days, I can be found at Invivo Research, Inc., where, among other duties, I develop software for an embedded system. I do this using a cross-development environment, where the host and target machine are both PC-based. Until recently, the host machine was a 50MHz 486-based PC clone. The machine runs Windows 3.1, and our software is developed in Microsoft Visual C and a bit of assembly language. I also use the host machine for a lot of analysis and simulation, using C and Mathcad.

The target machine uses a 66MHz 486. It has a normal PC motherboard, into which we've plugged our proprietary I/O boards and stuffed in the sensor hardware. Being an embedded system, the target machine normally doesn't have a disk drive, keyboard, or general-purpose serial ports. To support our development efforts, we rigged up a general-purpose I/O card to support a hard disk and two serial ports.

As is usually the case, space inside our embedded product is at a premium; not just any old I/O card would do. The card had to be trimmed on both ends to fit into a normally unused slot. We adapted one of those tiny, 2-in. drives from a laptop, fastening it directly onto the I/O card using double-backed foam tape.

When I'm developing software, I write the code in C, compile it on the host machine, and download it using LapLink. I debug using Winice, which allows me to see what's going on in the target machine while that one is busy drawing its graphics. It's not a perfect system; the version of Winice that we use is pretty primitive, thanks mostly to the fact that we're using it in a way that was never intended (with Phar Lap DOS extender). But it's enough to get the job done. If I really want to, I can even have my embedded software capture real-time data to a disk file (via a RAM buffer) and pipe it over to the host machine for analysis. Likewise, I can have it read test data from a file for use as simulated inputs.

Our host machines are, theoretically at least, connected to a net server via Novell NetWare, and, again, theoretically, the files are backed up onto mag tape by the server.

At home, until recently my desktop machine was also a 50MHz 486 Dell computer, with two hard disk drives totalling 750MB (1.5GB using Stacker), plus a Colorado tape backup system. I also ran Windows 3.1, and most of these articles have been written using Microsoft Word 2.0 and later, 6.0, with a little help from Mathcad, Corel Draw, Borland Quattro Pro, and other tools. These environments were pretty pleasant for me, and a lot of work got done using them.

BACKUP, BACKUP, BACKUP

We all know the value of, and need for, backing up important data. Despite the term "non-volatile memory," associated with disk drives, we know that the things can sometimes fail. However, I've noticed that my attitude towards backing up data has evolved over the years, along with the associated hardware. Perhaps yours has, too.

I've been using computers for a long

time, since shortly after the War Between the States. Nobody had to tell me about the need for frequent backups of my data.

Likewise, no one needed to tell me to save all my work on floppies—they were, after all, the only non-volatile media we had. When saving a file, it was easy enough to save it twice, to two different disks.

In those early days, most computers actually had *two* floppy drives, not the puny single drive that seems so normal today. So backing up data was as easy as typing copy a:*.* b: without having to do all that cursed disk-swapping that Microsoft has blessed us with. What's more, data files tended to run closer to 2K than the 200K that seems so normal today, so copying a file was fast—typically only one or two disk rotations.

I soon fell into a mode, which I still use today with tapes—when I have tapes—which is to alternate between two copies, so I always have backups two generations back. If a backup disk failed, or if I royally screwed it up by copying garbage onto it, I could always fall back to the previous generation and recover, losing no more than the data generated between backups.

TIME MARCHES ON

That was then, this is now. Back in the days when Wordstar came on one 8-in. floppy, making a backup copy was trivially easy. Nowadays, we have "progress," and applications that used to fill one floppy now require 20 or more. Making backup copies of these apps, with the single-floppy drive setup that seems to have become de rigeur, is now practically an all-day job, and a true exercise in masochism. Though I used to religiously make those copies of all the vendor's distribution disks, I've long since stopped the practice, and trusted their disks to last me through yet another re-installation. My desk drawers are already bulging with vendor's disks. Aside from the time required to copy them, the prospect of doubling their number is too depressing to contemplate. So I don't copy them anymore. How about you?

At least, with the setup I had until several months ago, I was still backing up data. My Dell 486 had two hard disk drives, so the first level of backup was simply to copy the most precious data to a mirror directory on the second drive. I figured, the probability of disaster happening to both drives at once was almost zero. I also had that Colorado tape backup system, which seemed to work quite well, though I must admit, there were times when the data I'd stored refused to reload. I quickly fell into my old practice, of maintaining two backup copies of both disks, two generations back. Finally, I continued, at least for a while, to keep the articles I've written on both floppy disks and hard copy. That should have been enough backup for even the most pessimistic believer in the power of Murphy's law. And it would have been, too, except that I began to be more lax.

I've been writing articles and columns like this one since 1988. At this point, I think I'm up to something like 200-plus articles. The logistics involved in making hard copies and floppy backups of each began to get out of hand. So by December of 1995, I had fallen back on relying upon the tape drive and the second hard drive, to keep things secure.

At work, I had no problem (I thought)—the automatic backups by the network should have protected me there. One slight problem: a couple of years ago, using Novell Netware with Windows 3.1, the performance of the system was less than thrilling. Netware seemed to be playing its own version of Russian roulette, randomly picking out workstations on the net to take down.

When Netware would take me down, I'd lose all the data in every open file. It didn't take too many of these experiences to convince me that I'd be better off not logged in, except when uploading or downloading data, or backing up to the network server. I thought this latter job was being done by the system administrator, once a week.

SOUNDS OF THUNDER

In December 1995, I decided to give myself a Christmas present. I bought a new 133MHz Pentium tower, complete with 1.6G Western Digital hard drive (no more need for Stacker), an 8x CD-ROM drive, and all the other usual bells and whistles like Sound Blaster, fast video card, multimedia support, and so on and so on.

I had only one small problem: The new computer was arriving with Windows 95 installed. Now, a colleague had told me that the Colorado tape system software was unreliable with Windows 95. He had had a bad experience in which he backed up all his data onto tape, then later discovered, to his horror, that the data wouldn't reload. As I was sitting in the dealer's office paying for my new toy, my eyes fell upon a new Colorado tape drive, capable of handling 1MB tapes. The labeling on the box proudly proclaimed, "Works with Windows 3.1 and Windows 95." Thinking to cut off any potential problems, I asked the dealer to give me that tape drive, also. Not only would it give me guaranteed Win 95 compatability, I reasoned, but I'd still have the old drive for the old computer, which I fully intended to retain. I'd put the new tape drive in myself.

Now, to complete the picture, I must report to you a fact that's well known to the *ESP* editors: I'm the world's worst procrastinator. I'm the founder and president of Procrastinator's Anonymous (we're going to have our first meeting, one of these days). I had that nice, new Colorado 1MB tape system, which I fully intended to install, some day Real Soon Now. Fifteen months later, it's still sitting here beside me in its box, in the "things to install" pile. What's more, the new system had only the single hard drive, so I could no longer copy files to a mirror directory on the second drive. I was running totally barefoot. I knew I was asking for trouble doing so, but hoped things would stay together a little while longer.

At work, changes were also under way. As of a few months ago, I was the only software developer still left using

CIRCLE # 9 ON READER SERVICE CARD

a 486 computer and Windows 3.1. Everyone else had upgraded to either Win 95 or NT. By this time, I was using Microsoft's SourceSafe to handle my configuration management, and that old 400MB drive was getting really crowded. I asked for and got another fancy Pentium system.

LIGHTNING STRIKES

By now, I'm sure you can see where all this is heading. Here I sat with a computer with a large, and seemingly super-reliable, hard disk. Backing my articles up onto floppies stopped long ago, thanks to both the sheer volume of the data, and the unreliability of the floppies. I got the impression that having copies on those cheap floppies (some of which, by the way, are brand name items) was almost worse than having none at all—it only gave the impression of security, not the reality. Ditto for the tape drive backup system which, according to my friend, seems to only work at certain phases of the moon. Though I had by no means lost my understanding that backups were essential, the pressure to back up seemed lessened by the fact that the backup media were, themselves, questionable.

Was I riding for a fall? No question about it. I might have well as stood on a mountaintop in the middle of a thunderstorm, and dared the lightning to strike. It did.

They say that trouble always strikes in threes, and so it seems in this case. About a month ago, I had just completed my latest article for *ESP*—at the last minute, as usual. I fired it off to my editor, Michael Shapiro, via Internet e-mail, and went to bed. I even did something unusual. As some kind of statement of finality, I imagine, I shut down the computer, CRT, laser printer, and even the modem. For the first time in 15 months, silence reigned in the computer room.

By now, I'm sure you can see where all this is heading . . . There I sat with a large, seemingly super-reliable, hard disk.

Two evenings later, I fired the system up again to talk on CompuServe. I was in the midst of answering a forum message, when my browser gave me an unusual error message: "Cannot find file xxx.dll." Hm, that's funny, I must have some broken directory chains (something that happens, in Windows, with depressing regularity). I got out of the reader, ran Scandisk, and found that, sure enough, some chains were broken. I fixed them, then tried again. Same result.

Thinking that somehow Windows had gotten its configuration settings messed up, I decided to power down the system and reboot. The news from the BIOS was even more ominous: "Non-system disk or disk error." Gack!

I tried resetting the master boot record on the hard drive. I tried reloading the operating system. Nothing helped. Inside of 15 or 20 minutes, the condition of the drive had gone from "can't open file" to "disk not present." The disk had degraded so badly that the system BIOS couldn't even tell there was a disk there. I had experienced the computer user's worst nightmare: a sudden, total, unrecoverable hard disk failure.

Fortunately, I did have a nice 3.2GB drive sitting in my "to-install" pile. I was soon up and running, but with no software, and worse yet, no data. The OS and applications I could reload, but 15 months' worth of articles, presentations, analyses, code, and so on, were gone forever. I could do nothing but sit there with this stunned, catatonic look on my face.

To make matters more interesting, Michael called me a couple of days later to tell me that the article I'd e-mailed to him had arrived in unreadable condition. Could I please re-send it? It took a lot of courage to explain to him why I couldn't—which is why you aren't reading about signed integer multiplication this month. You'll get that article, resurrected from my feverish brain, next month after I get this one off my chest.

AND AGAIN

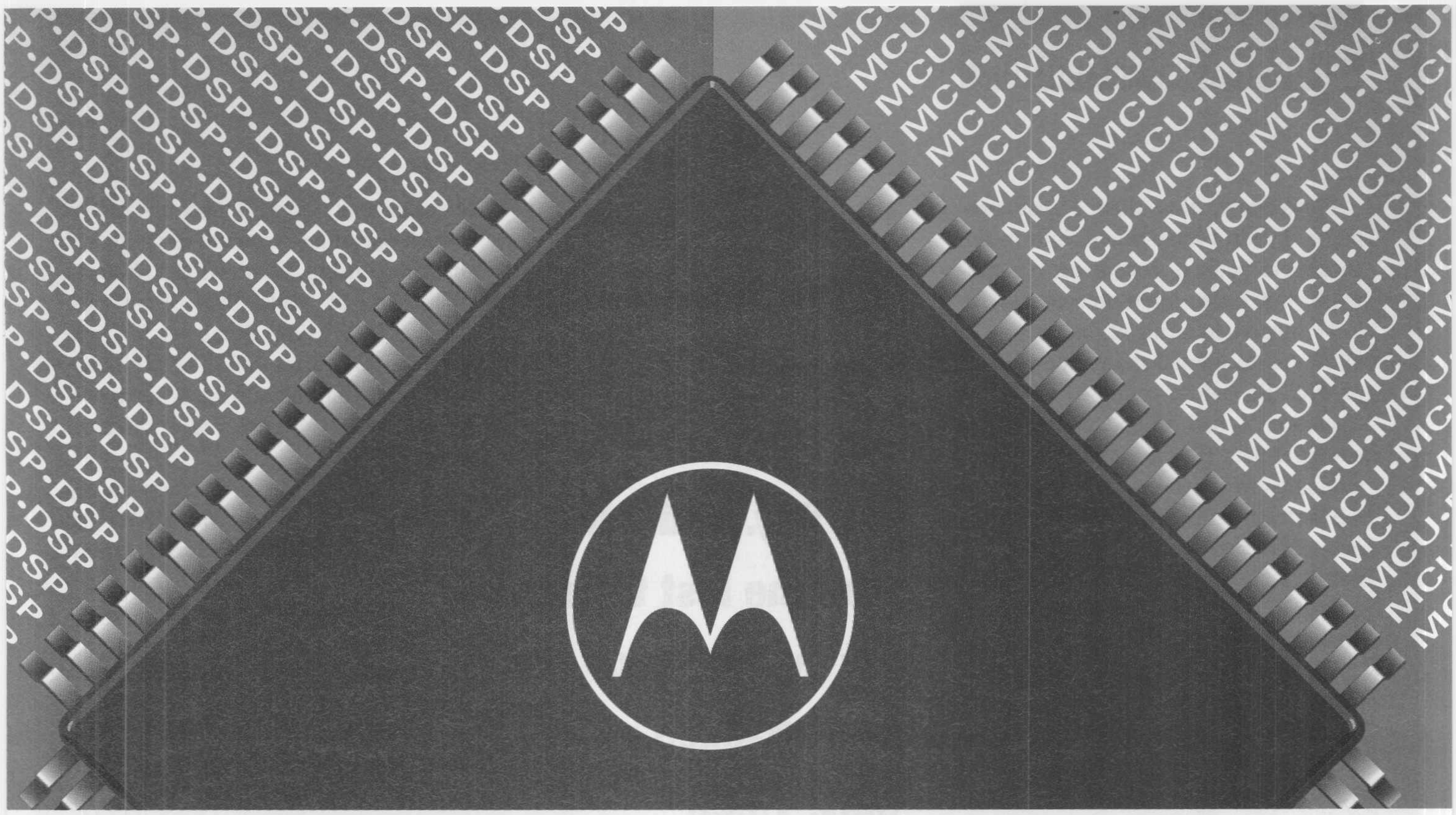
Meanwhile, over at work, my new, 150MHz Pentium system arrived. Of course, it arrived in a condition that I couldn't yet use to do useful work; it had Windows NT installed, but nothing else. I intended to remove the hard drive from the old computer, and connect it up as a slave drive for the new system. Then I'd just Xcopy all my files over.

Unfortunately, the system administrator, when he had installed NT, had elected the NTFS file system rather than the old FAT system. The new system could not read the old disk. It said it was reading it, and files seemed to be copying, but when I tried to use them, they didn't work.

Well, I wasn't sure I wanted NT anyhow. Most of my colleagues there have switched from Windows 3.1 to NT, but then back to Win 95 because they've found it to be more reliable as well as faster. In the course of my debugging, I have to go to DOS and back to Windows at each cycle. The boot process for NT is very slow, so I was persuaded by my friends to go the 95 route. So I got this bright idea: I'd reformat the new hard drive, swap the two drives, boot from the old one, and copy the files that way. Then I'd upgrade from Windows 3.1 to 95.

That's when I got my next depressing shock: "Non-system disk or disk error." Double gack!

What followed was a paroxysm of disk swapping, from the old to the new system in every possible configuration. I even tried LapLinking to the target machine and back, but by this time the old drive was corrupted to the point where LapLink wouldn't work, either.



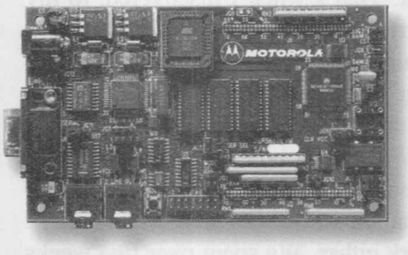
Our DSP56L811 has a split personality

(Good thing they're both so easy to get along with)

Is it a DSP? Is it a microcontroller? Well, it's both or it's either, the choice is yours. Microcontroller fans will see a general purpose, easy to program 16-bit controller, while DSP engineers can take advantage of a high-efficiency, low-power DSP architecture. The decision is easy.

Now you get a single part optimized for signal processing and microcontroller functions; plus it's easy to program, with compact assembly and C compiled code. This 3 V, 16-bit processor handles up to 20 MIPS @ 40MHz, and sells for less than $10 today. And whichever personality you like better, you won't need a lot of power, since the DSP56L811 typically consumes only 1.1 mA per MIPS.

Start your next design using the DSP56L811's robust array of development tools from both Motorola and independent developers. Our full featured DSP56L811 Evaluation Module not only includes software tools for application development, but it even comes with a codec and 128 Kbytes of SRAM already on board.



DSP56L811 Evaluation Module

So don't let the dual requirements of your next application drive you crazy. The DSP56L811 is the answer for cordless telephone, digital pager, digital tapeless answering machine, low speed modem and other consumer applications. Either way you look at it, increased functionality and performance at low cost is here to stay.

Call 1-800-845-MOTO for more information or to order your $149 evaluation module today. Or visit our website at http://www.motorola-dsp.com.

MOTOROLA

What you never thought possible.™

Motorola and Ⓜ are registered trademarks of Motorola, Inc.

Fortunately, this time I was prepared. I didn't have much faith in the automatic backup system on our network, but I also knew that it was far too crowded, in terms of disk space, to allow me to upload the entire 400MB of disk. So before trying to make the transition, I had zipped up my most essential data files and uploaded them manually to the network. Once I got the software reloaded back onto the new drive and hooked back up to the network, I downloaded these files, unzipped them again, and was back in business.

Unfortunately, I hadn't backed up every data file. There were some very important Mathcad documents, for example, that were stored in the Mathcad directory (its default directory) rather than where they should have been. Thinking I could always just reload the applications, I didn't include these files in the zip files I'd saved. Like my articles, they have now gone to that Great Bit Bucket in the Sky.

My next big shock came when I went to the system administrator. I explained that, although I'd managed to back up most of my data files, I'd missed a few, and needed to get copies from the server's archives. He pulled up the data from the last time my system had been backed up—July, 1995! More gacks! At this point, I'm now in the process of resurrecting these key files by re-entering the data directly for my databooks. Thank goodness that pencils and paper still work. As long as the temperature in Florida stays below 451° F, I guess the situation is still salvageable.

Among the data files in jeopardy were 50,000 lines of source code for our latest product—that's been nearly three years in development. To lose them would be a disaster. Fortunately, each time we make a new build, each developer uploads his/her code to the network server, and the complete build is then downloaded into the workstation of the person doing the build. Furthermore, to keep our own developments current, each developer periodically downloads the entire build to his/her own workstation. So we do have multiple, multiple copies of this precious code, and are probably OK, as long as we don't get a simultaneous failure on *all* the workstations—a possibility that, while remote, no longer seems unlikely to me.

Like my articles, these files have now gone to that Great Bit Bucket in the Sky. And when was the last time my system had been backed up? July, 1995!

AND YET AGAIN

As if two hard drive crashes weren't enough, shortly after I got the system's back up, the I/O board on my target machine failed. Because this is a customized board with a hard drive attached, replacing it is a non-trivial task. That's when I discovered, to my horror, that we had no spares in stock. What's more, the vendor no longer carried them, and they estimated a one month lag time to order one. Gack!

We found a board of similar shape in a surplus store. We trimmed and installed it, only to find that it didn't work either. We spent over two weeks looking for the problem. I'll spare you the tortures that we went through to find the trouble (this is a family magazine). But, believe it or not, we found the problem was actually an error in the board layout. No wonder it was surplus! We fixed the problem with a custom cable.

HAPPIER TOPICS

Some good things come out of this ordeal. First of all, when I got the new computer at work, I had to decide which operating system to use with it. I had been using Windows 3.1, both at home and at work. Two of my colleagues are using Windows 95, and another, Windows NT. The big question was, which should I use? Each person I talked to gave me a different answer.

Finally I posed the question on my favorite place to ask questions, the Software Development Forum (sponsored by Miller Freeman) on CompuServe. The answer I got was surprising: why not keep them all?

There are programs that let you do this, such as IBM's Boot Commander and System Commander by V Communications (which was highly recommended). With System Commander, you can actually maintain multiple OS's in the same partition of the same disk drive. Or, if you prefer, you can maintain multiple primary partitions (something DOS's FDISK won't allow), and put the OSs and their data in separate partitions. You can also have multiple configurations of the same OS. There is no practical limit, except your sanity, as to how many such systems you can use. System Commander modifies the disk's boot record, and loads itself before anything else happens. After you've selected the OS and configuration you want, System Commander executes the appropriate boot code and configuration files.

A wonderful companion program for System Commander is Partition Magic, from PowerQuest. Partition Magic is aptly named, because it allows you to do something I wouldn't have thought possible—it allows you to create, delete, move, and/or resize disk partitions on the fly, without losing any data. Using DOS's FDISK, as you know, such things can only be done by reformatting the entire disk. If, after you've loaded your software, you realize you made the partitions wrong, in size or number, your only recourse is to back everything up and start over.

Partition Magic is a lifesaver because if you foul up and get the partitions wrong, it only takes a few seconds to fix the problem. As a neat side effect, Partition Magic saves disk space, because it will not only partition on the fly, but also redo the FAT tables for the optimal cluster size. Amazing!

I should tell you that I don't recommend software packages lightly. Come to think it, I don't even *buy* them lightly. You can appreciate, then, that if I'm recommending these two products as a team, they must work flawlessly together, and they do. As I type this, both my home and office computers are merrily humming away with three primary partitions on drive C:, one for DOS/Window 3.1, one for Windows 95, and one for Windows NT. Should I decide to do so in the future, I can still install Linux, OS/2, or any other of a number of operating systems, without disturbing the existing ones in the slightest. I like that.

THE CASUALTY OF WAR

You may be wondering what happened to that hard drive that crashed so suddenly, taking so much irreplaceable data with it. Well, that's a story with a bittersweet ending. I knew that there were companies that specialize in trying to recover erased, encrypted, or otherwise corrupted data from hard drives. I found one such company nearby, Data Recovery Labs, Inc. (DRL) in Clearwater, Florida. They claim an 85% success rate in recovering data, and my sources on CompuServe tell me they're really even better than that.

Here's how it works. Most disk failures are caused by something simple: A chip burns out, a read/write head fails, or the motor fails. In such cases, DRL can replace the offending part and get your drive back on line, at least long enough to recover the data. The data can be transferred to another hard drive, a CD-ROM, or any other medium you choose. To do the work they have to take the drive apart. This kind of thing requires clean-room practices, but DRL has just such a clean room, and makes such repairs routinely.

If your disk drive is under warranty, as mine was, DRL will order a replacement drive from the vendor. They will then swap out the parts from the new drive to the old. Then they order yet another replacement drive, and transfer your data to the new drive. You end up with a brand new drive with all your data securely in place.

The estimate I was given for this service was $300 to $800, which might seem high, but when you consider the value of all that data I lost, it would have been well worth it.

Unfortunately, things didn't turn out so well in my case. My drive was a Western Digital Caviar 31600, a 1.6MB drive. It had what DRL described as "the classic Western Digital head crash." It seems that there was a flaw in the design of this particular drive that allows the platters to warp with heat. This allows the head, which is supposed to fly above the surface on a cushion of air, to contact the disk surface. This surface contains the iron oxide medium, protected by a thin plating of chromium. The disk will actually run this way for awhile but eventually the head wears through the chromium layer and starts rubbing on the iron oxide itself. Iron oxide is abrasive, and acts like sandpaper inside the drive. In a short time, the head wears completely through the oxide layer.

This was the condition of my drive, according to DRL (who will supply photos as proof, if anyone doubts their story). There was no way to read the data on the bottom sides of the platters, because there was no oxide layer left for the data to reside on. All those precious documents were reduced to a brown powder clogging the air filter on the drive.

Since this incident, I've talked to some other friends on CompuServe who have heard similar stories. It seems that this problem only occurred with the Caviar 31600 model, not the entire line. Western Digital has since discontinued the 31600 and is offering the 2GB drive instead. The failure rate of this drive is normal, I'm told.

In the end, I turned out to be part of the 15% who DRL couldn't help. I did get a break on the price—they charged me only $160.

So, the good news out of all this is, there are recourses for people like me who are too busy or too dumb to perform periodic backups. Companies like DRL can often recover the data. The bad news is, they couldn't do it in my case. The good news is, they didn't charge me much, plus I got a brand-new, replacement drive, which is now in place in my home system, serving as the mirror drive.

The bad news is, it's another Caviar 31600.

LESSONS LEARNED

After all these problems, what have I learned? Here are some important lessons, some of which, of course, are obvious:

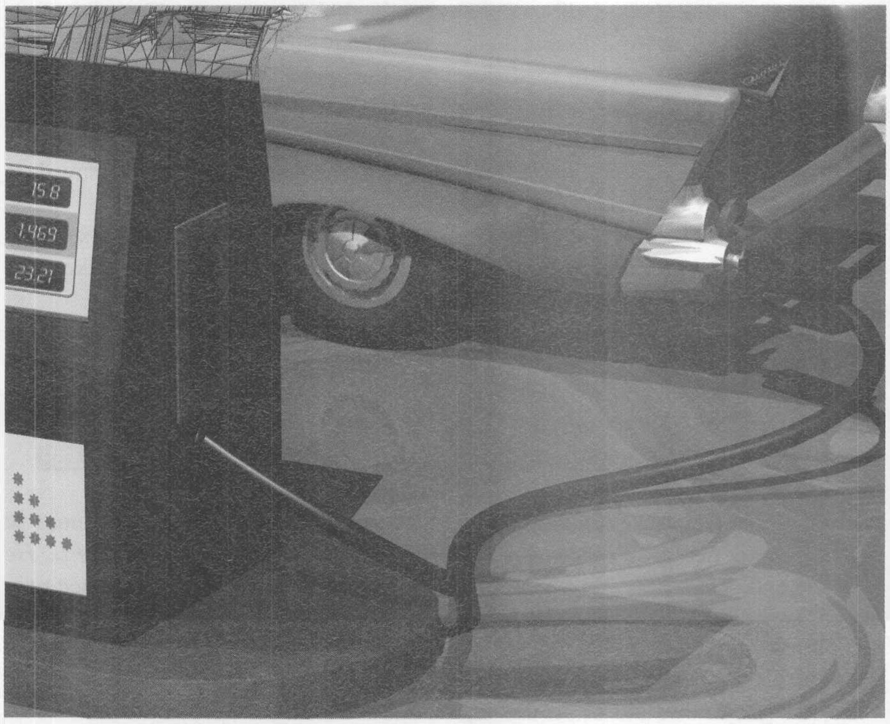
- Always back up data files daily. The backup medium isn't as important as the act. Back up to floppies, tape, disk, network, or Zip drive, but back up. Losing irreplaceable data is no fun
- If possible, use two hard drives, using one as a mirror backup for the other
- Use at least two backups, stored in separate physical locations
- If you're on a network, make sure the files on your workstation are really being backed up. Better yet, back them up yourself
- Make sure the network server is being backed up, as well
- Never buy hardware for an important project from a discount or surplus house. The money saved is simply not worth the hassle
- For specialized hardware such as our custom I/O cards, make sure spares are always on hand
- When sending critical data via e-mail, send it twice, preferably to different people
- Don't ever buy a Western Digital Caviar 31600, at any price **ESP**

Jack Crenshaw is a staff scientist at Invivo Research in Orlando, FL. He did much early work in the space program and has developed numerous analysis and real-time programs. Crenshaw can be reached at 72325.1327@compuserve.com.

by BRAD HUNTING

Finite Word Length Effects on Digital Filter Implementations

Some forms of digital filters are more appropriate than others when real-world effects are considered. This article looks at the effects of finite word length and suggests that some implementation forms are less susceptible to the errors that finite word length effects introduce.



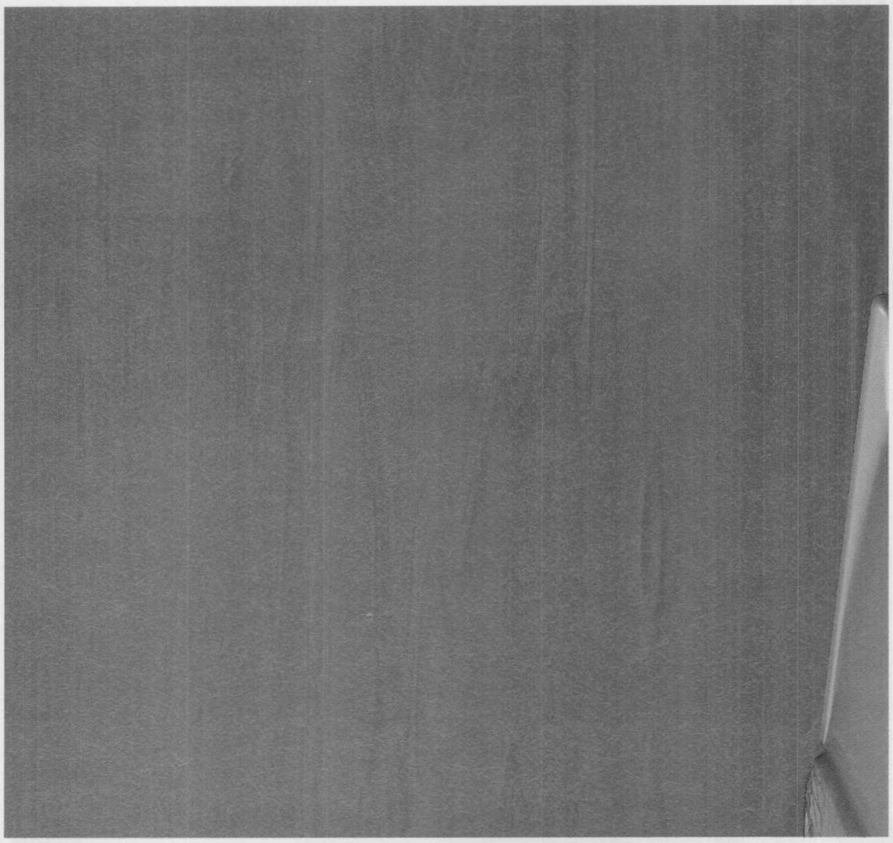
Rupert Adley

In articles about digital signal processing (DSP) and digital filter design, one thing I've noticed is that after an in-depth development of the filter design, the implementation is often just given a passing nod. References abound concerning digital filter design, but surprisingly few deal with implementation. The implementation of a digital filter can take many forms. Some forms are more appropriate than others when various real-world effects are considered. This article examines the effects of finite word length. It suggests that certain implementation forms are less susceptible than others to the errors introduced by finite word length effects.

FINITE WORD LENGTH

Most digital filter design techniques are really discrete time filter design techniques. What's the difference? Discrete time signal processing theory assumes discretization of the time axis only. Digital signal processing is discretization on the time and amplitude axis. The theory for discrete time signal processing is well developed and can be handled with deterministic linear models. Digital signal processing, on the other hand, requires the use of stochastic and nonlinear models. In discrete time signal processing, the amplitude of the signal is assumed to be a continuous value—that is, the amplitude can be any number accurate to infinite precision. When a digital filter design is moved from theory to implementation, it is typically implemented on a digital computer. Implementation on a computer means quantization in time and amplitude—which is true digital signal processing.

Digital filters often need to have real-time performance—that usually requires fixed-point integer arithmetic.



Computers implement real values in a finite number of bits. Even floating-point numbers in a computer are implemented with finite precision—a finite number of bits and a finite word length. Floating-point numbers have finite precision, but dynamic scaling afforded by the floating point reduces the effects of finite precision. Digital filters often need to have real-time performance—that usually requires fixed-point integer arithmetic. With fixed-point implementations there is one word size, typically dictated by the machine architecture.

Most modern computers store numbers in two's complement form. Any real number can be represented in two's complement form to infinite precision, as in Equation 1:

$$x = X_m\left(-b_0 + \sum_{i=1}^{\infty} b_i 2^{-i}\right) \qquad (1)$$

where b_i is zero or one and X_m is scale factor. If the series is truncated to $B+1$ bits, where b_0 is a sign bit, there is an error between the desired number and the truncated number. The series is truncated by replacing the infinity sign in the summation with B, the number of bits in the fixed-point word. The truncated series is no longer able to represent an arbitrary number—the series will have an error equal to the part of the series discarded. The statistics of the error depend on how the last bit value is determined, either by truncation or rounding.

COEFFICIENT QUANTIZATION

The design of a digital filter by whatever method will eventually lead to an equation that can be expressed in the form of Equation 2:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}} \qquad (2)$$

with a set of numerator polynomial coefficients b_i, and denominator polynomial coefficients a_i.

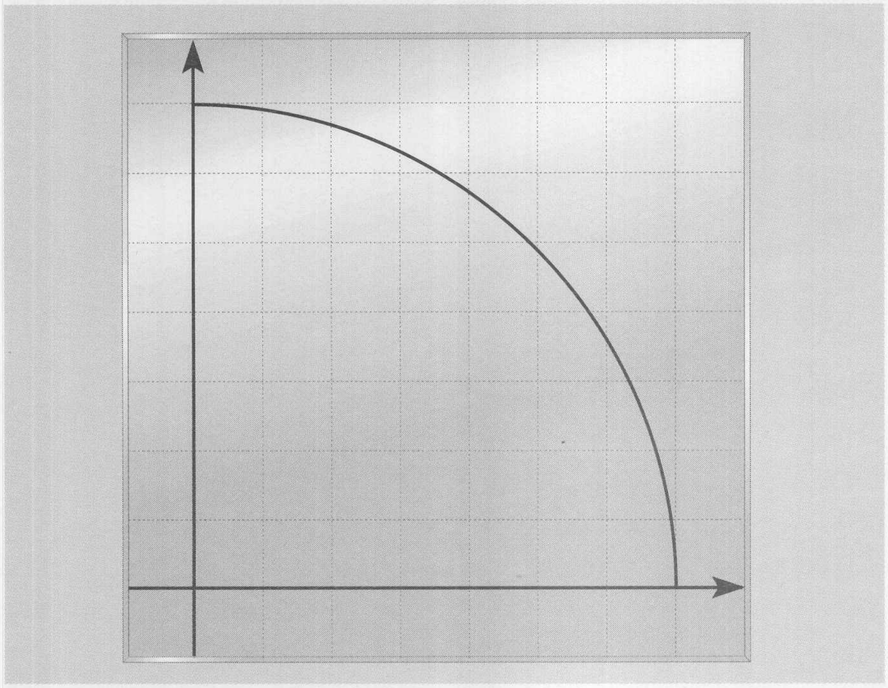
When the coefficients are stored in the computer, they must be truncated to some finite precision. The coefficients must be quantized to the bit length of the word size used in the digital implementation. This truncation or quantization can lead to problems in the filter implementation.

The roots of the numerator polynomial are the zeroes of the system and the roots of the denominator polynomial are the poles of the system. When the coefficients are quantized, the effect is to constrain the allowable pole zero locations in the complex plane. If the coefficients are quantized, they will be forced to lie on a grid of points similar to those in Figure 1.

If the grid points do not lie exactly on the desired infinite precision pole and zero locations, then there is an error in the implementation. The greater the number of bits used in the implementation, the finer the grid and the smaller the error.

So what are the implications of forcing the pole zero locations to quantized positions? If the quantization is coarse enough, the poles can be moved such that the performance of the filter is seriously degraded, possibly even to the point of causing the filter to become unstable. This condition will be demonstrated later.

FIGURE 1
Complex plane possible pole zero locations with finite word length.



ROUNDING NOISE

When a signal is sampled or a calculation in the computer is performed, the results must be placed in a register or memory location of fixed bit length. Rounding the value to the required size introduces an error in the sampling or calculation equal to the value of the lost bits, creating a nonlinear effect. Typically, rounding error is modeled as a normally distributed noise injected at the point of rounding. This model is linear and allows the noise effects to be analyzed with linear theory, something we can handle. The noise due to rounding is assumed to have a mean value equal to zero and a variance given in Equation 3:

$$\sigma_B{}^2 = \frac{2^{-2B}}{12} \qquad (3)$$

For a derivation of this result, see *Discrete Time Signal Processing.*[1] Truncating the value (rounding down) produces slightly different statistics. Multiplying two *B*-bit variables results in a 2*B*-bit result. This 2*B*-bit result must be rounded and stored into a *B*-bit length storage location. This rounding occurs at every multiplication point.

SCALING

We don't often think about scaling when using floating-point calculations because the computer scales the values dynamically. Scaling becomes an issue when using fixed-point arithmetic where calculations would cause over- or under flow. In a filter with multiple stages, or more than a few coefficients, calculations can easily overflow the word length. Scaling is required to prevent over- and under flow and, if placed strategically, can also help offset some of the effects of quantization.

SIGNAL FLOW GRAPHS

Signal flow graphs, a variation on block diagrams, give a slightly more compact notation. A signal flow graph has nodes and branches.

The examples shown here will use a node as a summing junction and a branch as a gain. All inputs into a node are summed, while any signal through a branch is scaled by the gain along the branch. If a branch contains a delay element, it's noted by a $z - 1$ branch gain. Figure 2 is an example of the basic elements of a signal flow graph. Equation 4 results from the signal flow graph in Figure 2:

$$C(z)=e\big(dB(z)+z^{-1}A(z)\big) \tag{4}$$

FIGURE 2
Basic elements of a signal flow graph.

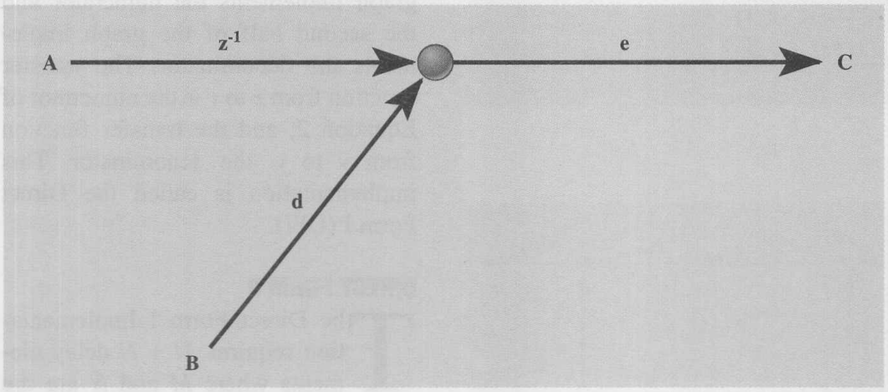


FIGURE 3
Direct form I signal flow graph.

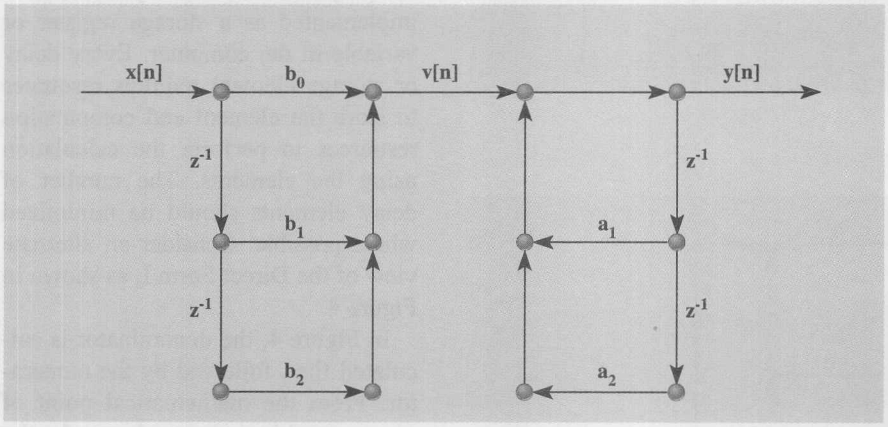
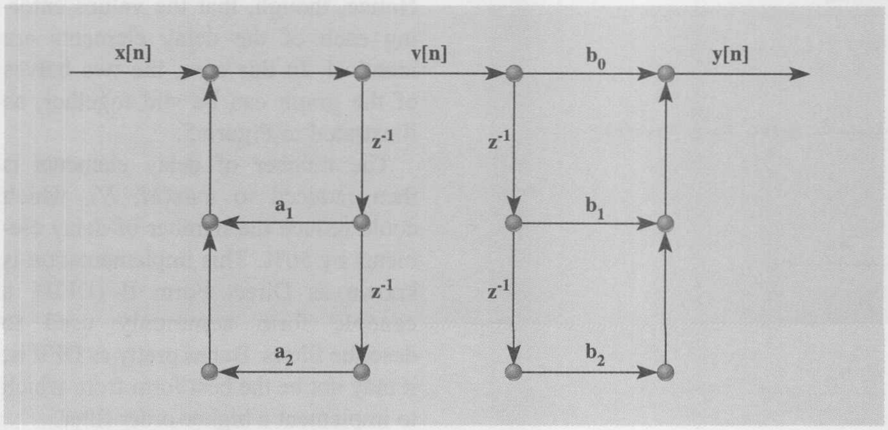


FIGURE 4
Direct form I alternate configuration.



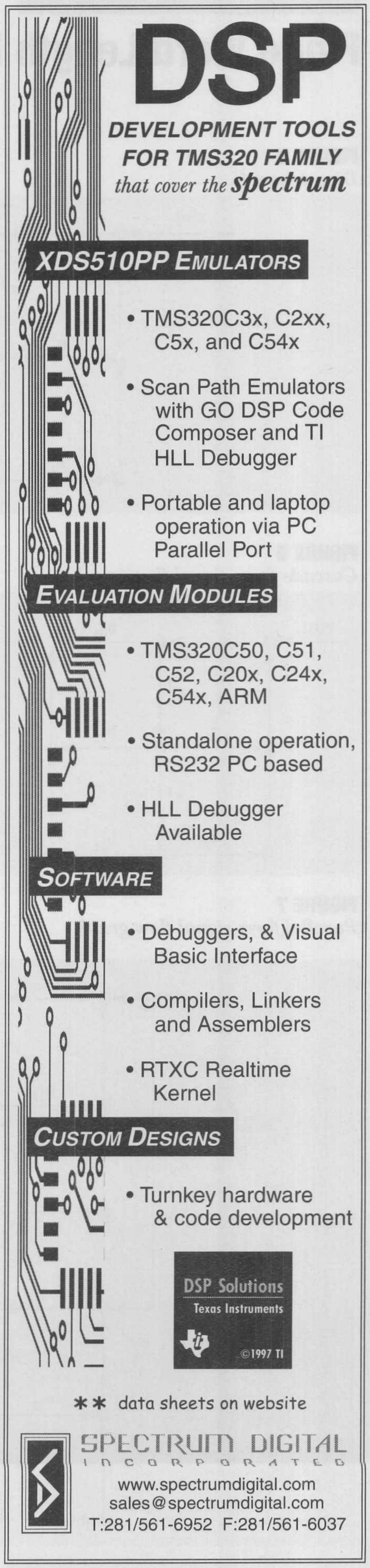


FIGURE 5
Direct form II.

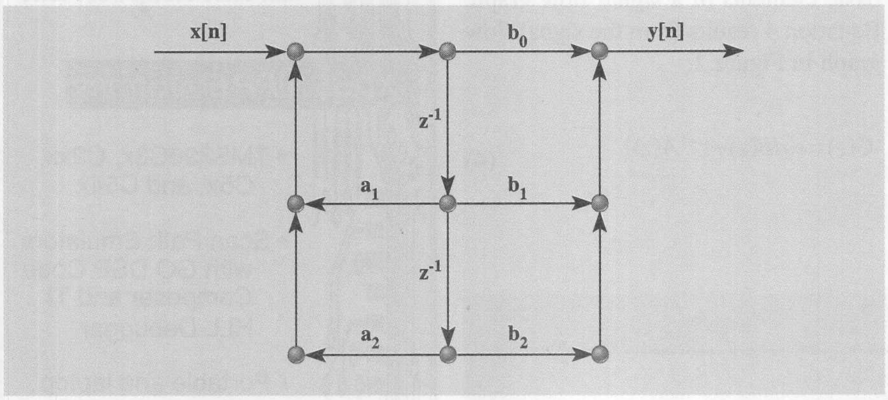


FIGURE 6
Cascade form signal flow graph.

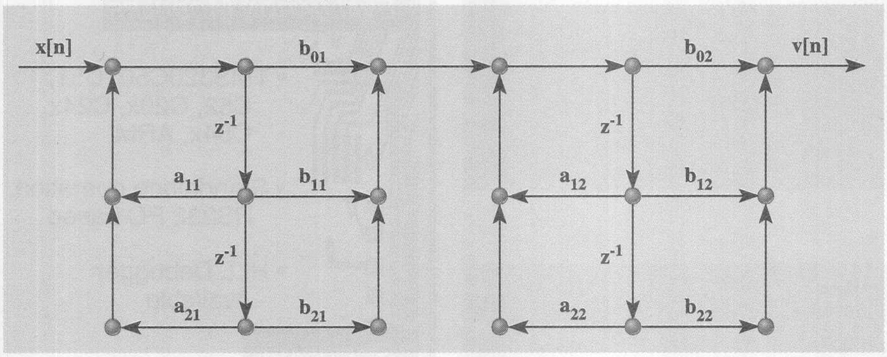
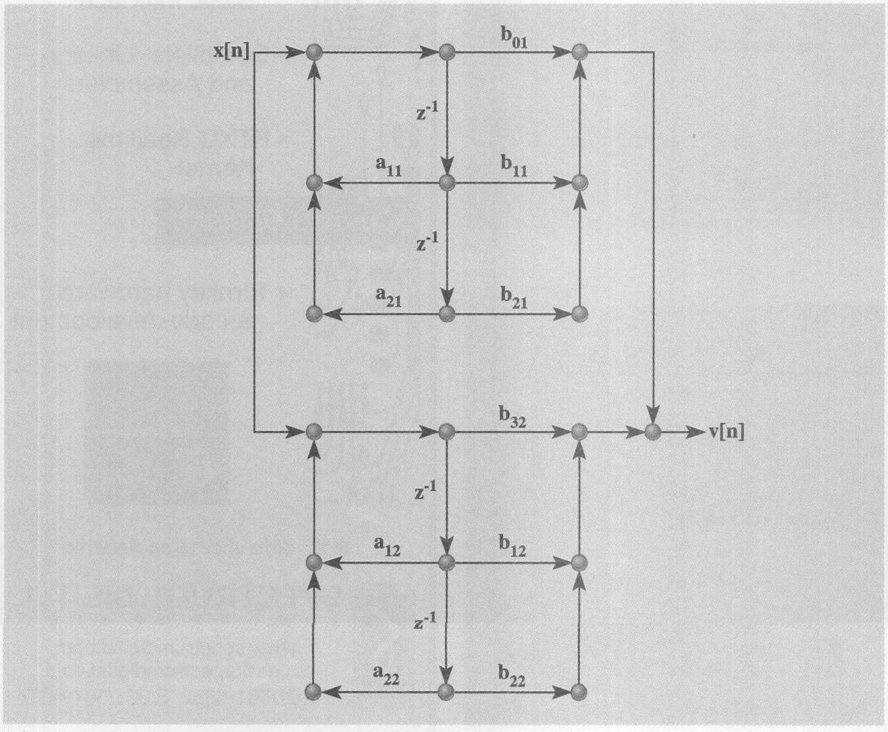


FIGURE 7
Parallel form signal flow graph.



DIRECT FORM I

A filter design will generally yield a set of filter coefficients which describe a filter as a polynomial numerator over a polynomial denominator, similar to Equation 2. The direct implementation of Equation 2 as a signal flow graph is shown in Figure 3.

The first half of the signal flow graph implements the numerator and the second half of the graph implements the denominator. The transfer function from x to v is the numerator of Equation 2, and the transfer function from v to y, the denominator. This implementation is called the Direct Form I (DFI).

DIRECT FORM II

The Direct Form I implementation requires $M + N$ delay elements where M and N are the orders of the numerator and denominator polynomials. A delay element is implemented as a storage register or variable in the computer. Every delay or storage element requires resources to store the element and computation resources to perform the calculation using the elements. The number of delay elements should be minimized where possible. Consider an alternate view of the Direct Form I, as shown in Figure 4.

In Figure 4, the denominator is calculated first, followed by the numerator. From the mathematical point of view, nothing has changed—the results of the calculation are the same. Notice, though, that the values entering each of the delay elements are identical. In this case, the two halves of the graph can be slid together, as illustrated in Figure 5.

The number of delay elements is then reduced to max(M, N), which could reduce the number of delay elements by 50%. This implementation is known as Direct Form II (DFII), a canonic form commonly used to describe filters. But as pretty as DFII is, it may not be the best form from which to implement a higher-order filter.

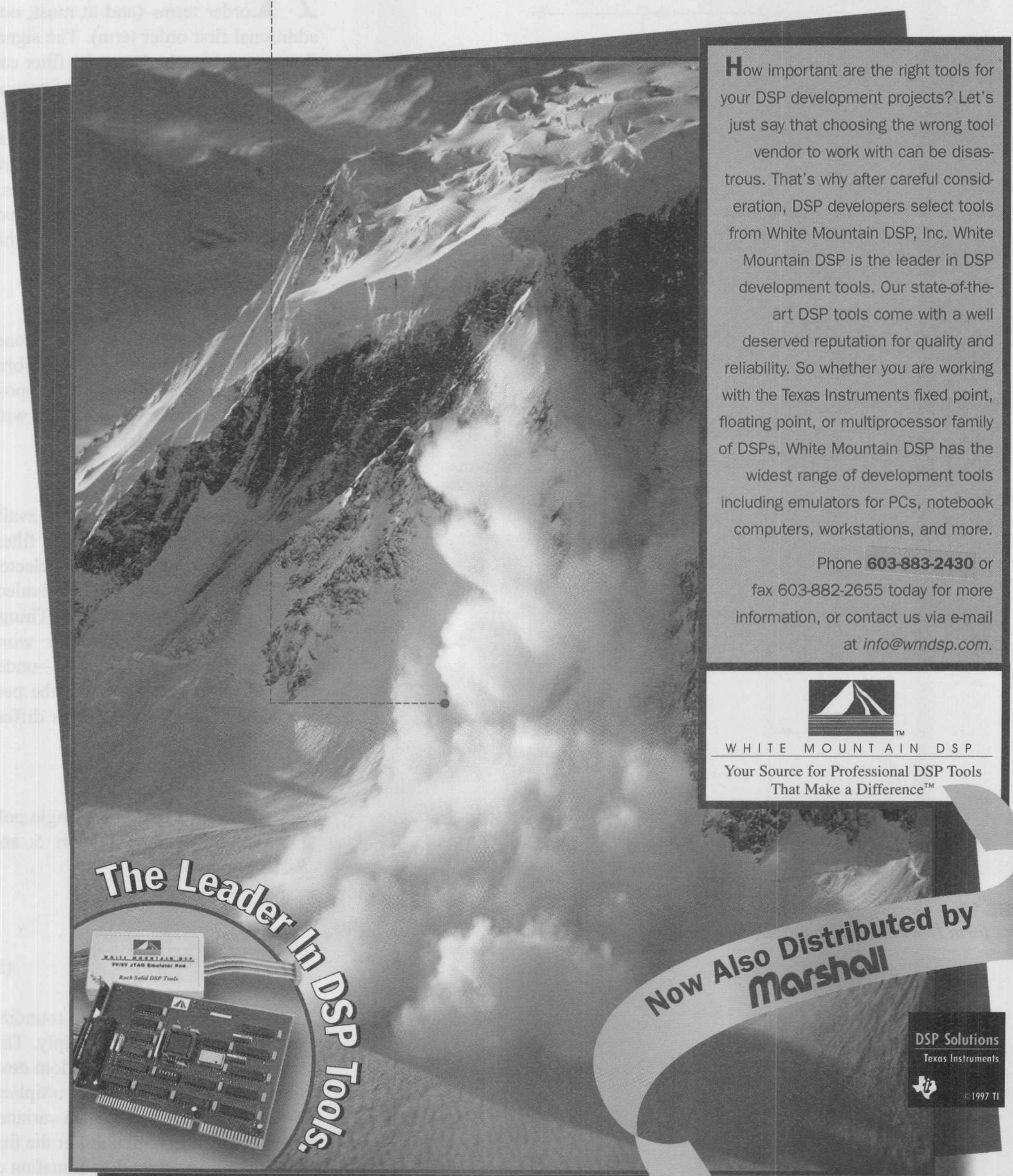

The Leader In DSP Tools.
White Mountain DSP
5V/3V JTAG Emulator Pod
Rock Solid DSP Tools
Now Also Distributed by Marshall
DSP Solutions
Texas Instruments
©1997 TI

FIGURE 8
First order filter signal flow graph.

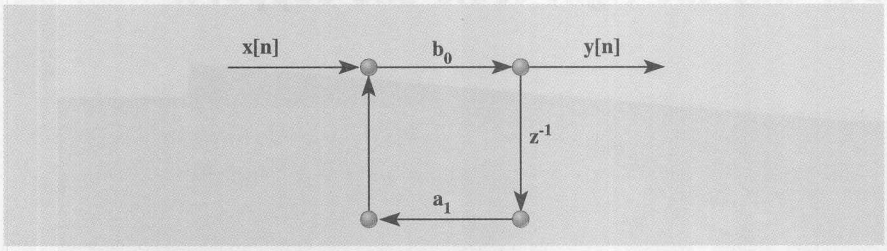


FIGURE 9
First order filter with linearized noise.

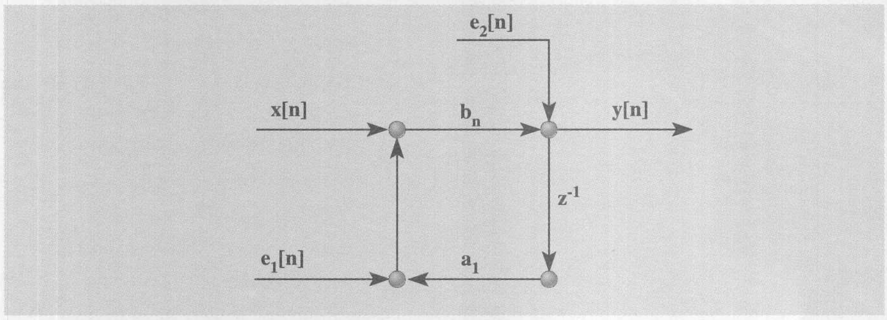


FIGURE 10
Direct Form I with rounding error sources.

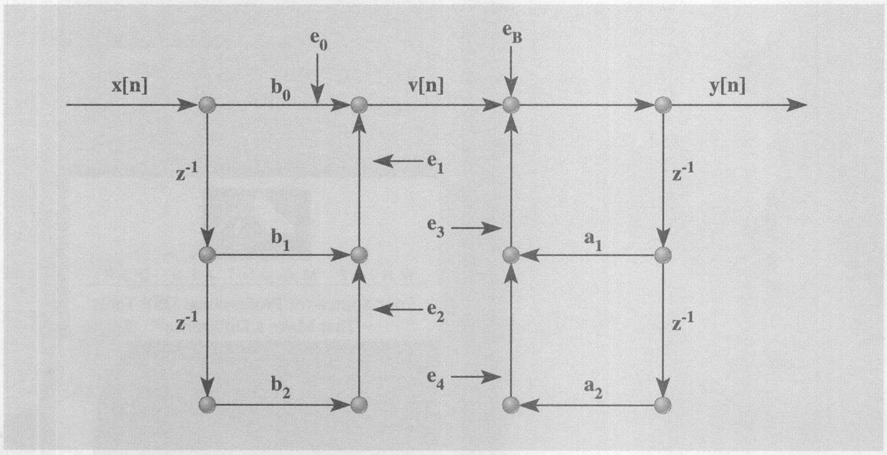
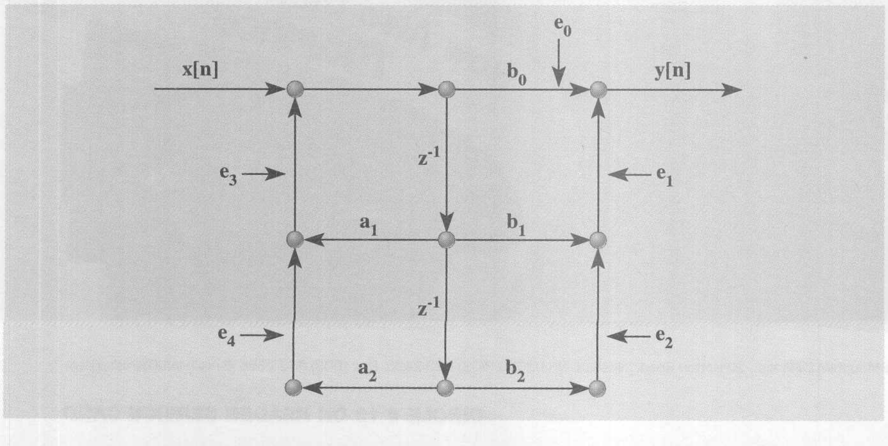


FIGURE 11
Direct form II with rounding error sources.



CASCADE

Any polynomial can be represented as a product of second order terms (and at most, one additional first order term). The signal flow graph for a higher order filter can be decomposed into a series of second order DFII sections, as illustrated in Figure 6. This is the form of the equation recommended by Jack Crenshaw in his *Programmer's Toolbox* series.[2] When we're through here, you'll have a better understanding of why this form is best.

PARALLEL

The parallel form is the final one, shown in Figure 7. This form can be arrived at by decomposing a higher order polynomial with partial fraction expansion.

IMPLEMENTATIONS

Given the different forms available to describe a digital filter, why would one be selected over another? The forms are equivalent under continuous math. Things change, though, when finite word length arithmetic is considered—under finite word length arithmetic, the performance of the various forms differs significantly.

NOISE MODELS

Consider a first order single pole filter, as in Equation 5 and Figure 8:

$$H(z)=\frac{Y(z)}{X(z)}=\frac{b_0}{1+a_1z^{-1}} \tag{5}$$

As I mentioned previously, a rounding error occurs after every multiply. This rounding is modeled as a random error injected at the output of the multiplication with a mean of zero and a variance that is given in Equation 3. For the first order system described in Equation 5, the linear noise model of the system is shown in Figure 9. The output equation for this system becomes as follows (Equation 6):

Mr. Sulu, I think you dropped something.

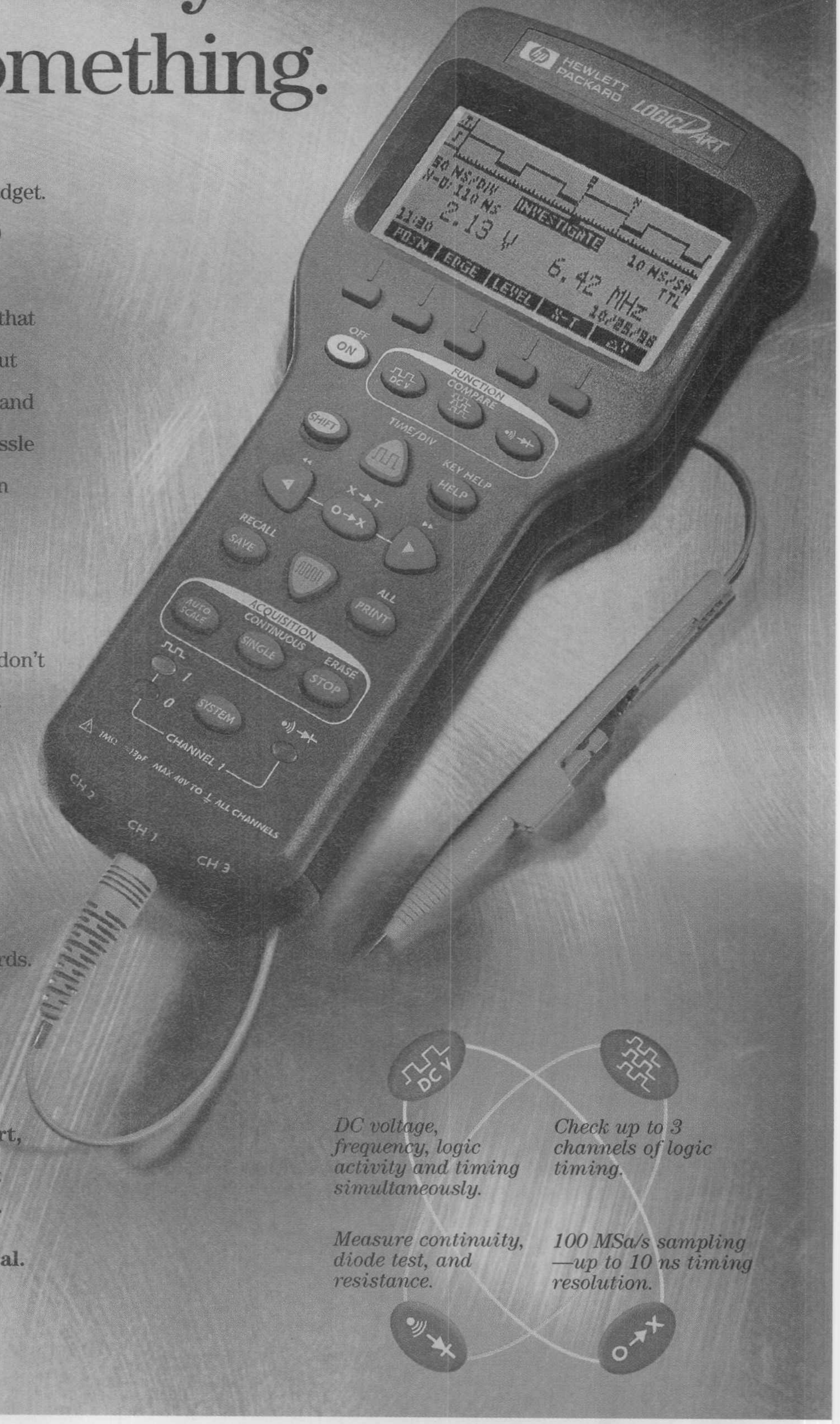
We admit the tri-corder is a pretty handy gadget. Unfortunately, you'll have to wait about 200 years to get one. Until then, consider the HP LogicDart. It's an advanced logic probe that not only performs basic logic monitoring, but also tests continuity, dc voltage, frequency, and does timing analysis. This eliminates the hassle of switching tools and keeps you focused on solving problems. Making it the perfect tool for first-level troubleshooting.

What's more, you can make all of these measurements from a single probe. So you don't have to fumble around hooking up different probes for different instruments. And the HP LogicDart's precise probe tip allows you to confidently probe fine-pitch surface mount circuitry.

But there's more to this 23rd-century tool than we can explain in 150 20th-century words. That's why we're offering a free brochure with key specifications and ten practical HP LogicDart applications.

To find out more about the HP LogicDart, visit our Web site or call HP DIRECT at 1-800-452-4844, Ext. 5217. At a price of only $795*, anything else would be illogical.

http://www.hp.com/info/LogicDart1



DC voltage, frequency, logic activity and timing simultaneously.

Check up to 3 channels of logic timing.

Measure continuity, diode test, and resistance.

100 MSa/s sampling —up to 10 ns timing resolution.

* U.S. list price Faxback: 1-800-800-5281, Document 12687

©1997 Hewlett-Packard Co. TMPMO533/ESP

HEWLETT® PACKARD

CIRCLE # 13 ON READER SERVICE CARD

$$Y(z)=\frac{b_0}{1+a_1z^{-1}}X(z)+\frac{b_0}{1+a_1z^{-1}}E_1(z)+E_2(z) \qquad (6)$$

The noise injected at E_1 passes through the system and is filtered as if it were noise injected at the input. The noise injected at E_2 is coupled to the output and passes directly to the output.

Given our new knowledge of noise models, we can now look at the higher order implementation forms. We'll consider only two forms, for lack of space. First consider the Direct Form I. Figure 10 shows the model for a second order DFI filter with linear noise sources modeled at the output of each of the five multiplies.

The noise sources E_0 through E_4 can be considered separately or combined at E_B. The direct path from each noise source to the output yields the noise contribution to the output signal with a mean of zero and a variance given in Equation 7:

$$\sigma_e{}^2=(M+N+1)\frac{2^{-2B}}{12} \qquad (7)$$

where M is the order of the numerator polynomial and N is the order of the denominator polynomial.

Consider a Direct Form II implementation. Figure 11 shows a DFII implementation with noise sources modeled. The noise contribution to the output is given in Equation 8:

$$\sigma_e{}^2=N\frac{2^{-2B}}{12}\sum_{n=-\infty}^{\infty}|h[n]|^2+(M+1)\frac{2^{-2B}}{12} \qquad (8)$$

where $h[n]$ is the filter impulse response. The noise generated by the zero calculations is still directly coupled to the output, and the noise generated by the pole calculations is filtered by the entire system. A DFII implementation will not necessarily have a lower noise output than a DFI, but it's true that part of the noise is filtered by the system in a DFII implementation.

TABLE 1
Tenth order Chebyshev digital filter coefficients.

Numerator Coefficients	x 10^{-3}	Denominator Coefficients	
b0	0.00066103030716	a0	1.00000000000000
b1	0.00661030307203	a1	-7.30444416597652
b2	0.02974636381481	a2	25.20380066635336
b3	0.07932363691054	a3	-53.82690864944371
b4	0.13881636441226	a4	78.53835694927194
b5	0.16657963752209	a5	-81.62627938641144
b6	0.13881636439805	a6	61.10920782789896
b7	0.07932363691765	a7	-32.51177043797594
b8	0.02974636379882	a8	11.75985136569655
b9	0.00661030307647	a9	-2.61175717587389
b10	0.00066103030666	a10	0.27062773957031

FIGURE 12
Tenth order Chebyshev digital filter response.

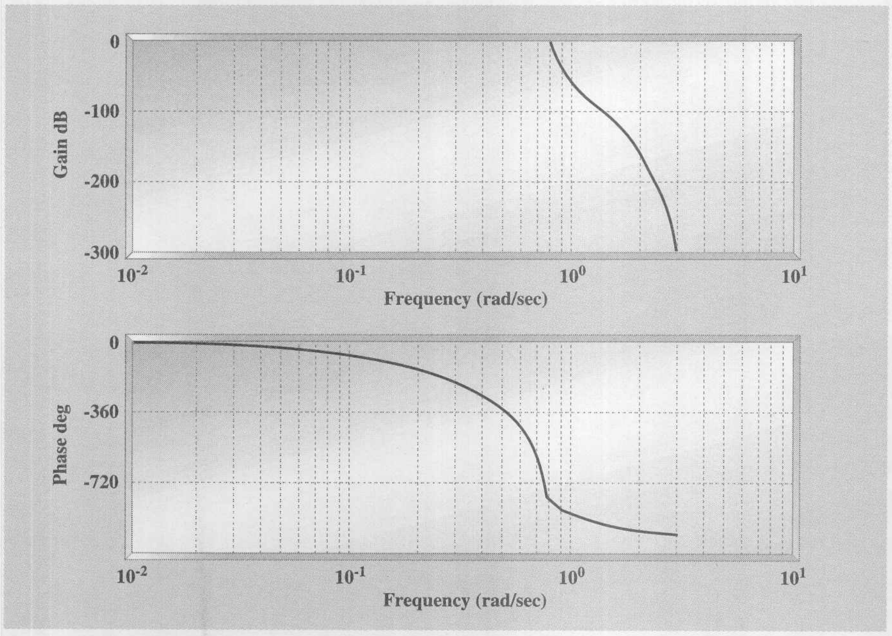
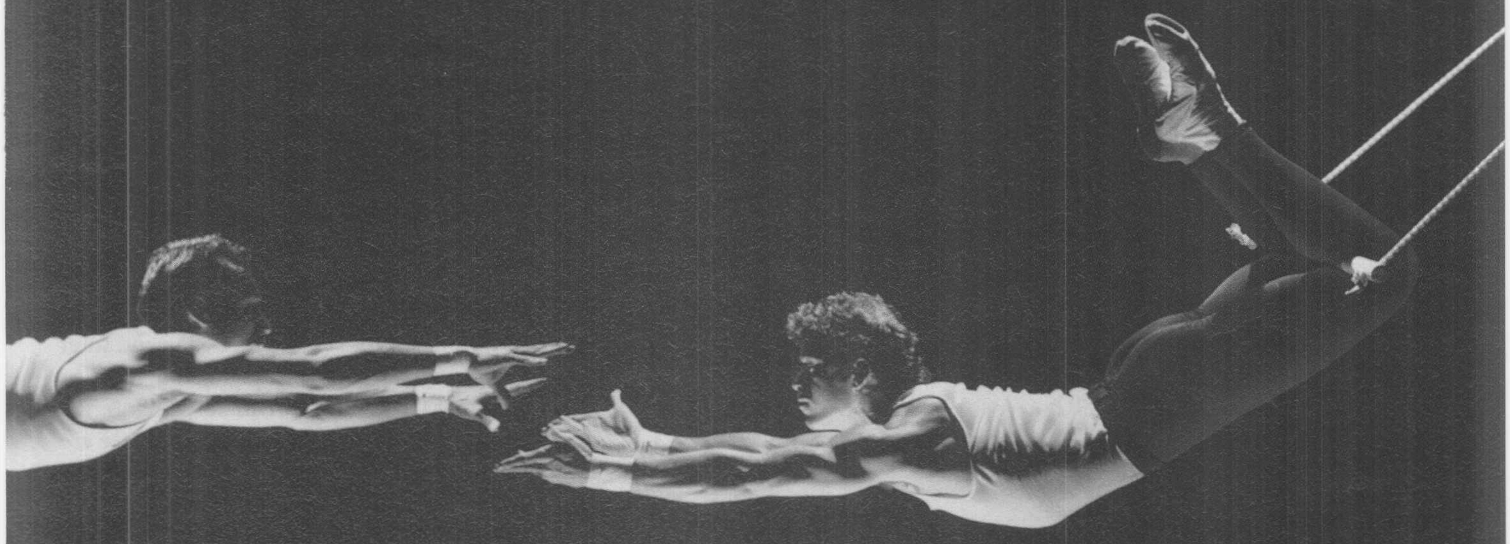


TABLE 2
Filter root locations and magnitudes.

Roots of Unquantized Poles	Magnitude	Roots of Quantized Poles	Magnitude
0.67689787328399 +/- 0.69719981309634i	0.97173983670423	0.68236236358186 +/- 0.73922698390391i	1.00601934820594
0.68392229911696 +/- 0.61223927931302i	0.91792518560238	0.92109158298132 +/- 0.50984979400459i	1.05278512370073
0.72193038646088 +/- 0.48422408315146i	0.86928501977167	0.59539467311044 +/- 0.61297725366473i	0.85453843113030
0.76884440604023 +/- 0.31357503601446i	0.83033187576463	0.85173230339742 +/- 0.15163783000507i	0.86512539445986
0.80062711808620 +/- 0.10895424215805i	0.80800668877135	0.60141907692895 +/- 0.28342414824416i	0.66485649120844

Real-time success is measured by flawless performance.



Nail it with INtime™ for Windows NT®

INtime™

REAL-TIME FOR WINDOWS NT®

You've set the stage with Windows NT,® but your real-time application demands precision. You need INtime™–the real-time extension to Windows NT from RadiSys.®

When determinism is critical. INtime's proven real-time technology is already successfully managing thousands of critical applications worldwide. With INtime's robust design, your real-time processes will survive even system software failures and will continue providing unshakable performance. INtime lets you exercise complete control over your real-time tasks–its microsecond response assures you of reliable, predictable behavior.

Work smarter. You no longer need proprietary or specialized operating systems to get hard real-time performance. With INtime, you can take advantage of all the things that have made Windows NT the industry standard for reliable computing–world class development tools, universal hardware support, and a wealth of third-party software. INtime's development wizards are fully integrated into Windows NT, so you'll quickly be up-to-speed developing real-time applications.

We're in the right place at the right time. RadiSys has what it takes to be your real-time partner. To get started with Windows NT and INtime, call us at **1-888-4REALNT**, or visit our web site at **www.radisys.com/intime/info**. Your success is on the line. INtime is your perfect catch.

REGISTER ON-LINE FOR FREE WEB SOFTWARE.

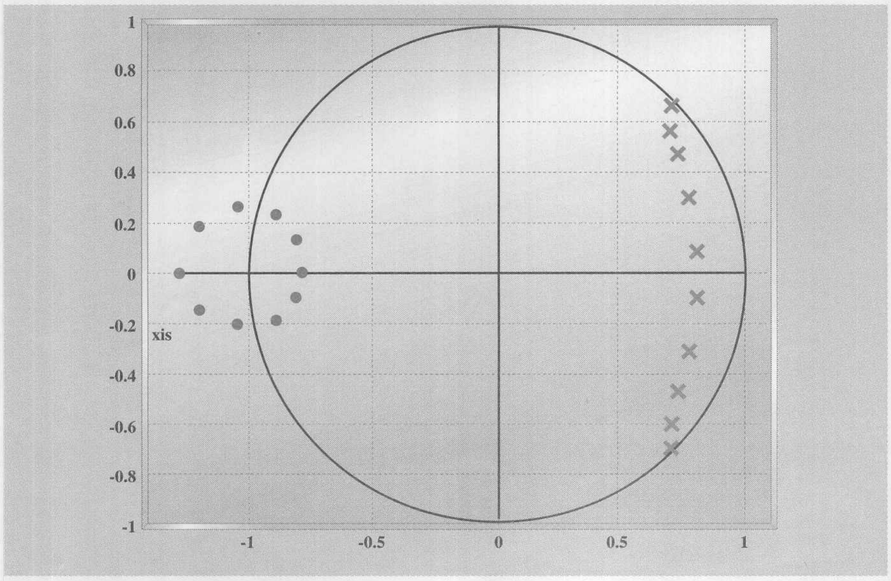
RadiSys.

Embedded Computer Solutions

Windows NT is a registered trademark of Microsoft Corporation.
RadiSys is a registered trademark and INtime is a trademark of RadiSys Corporation.

CIRCLE # 14 ON READER SERVICE CARD

FIGURE 13
Pole zero plot of double precision tenth order filter.



EQUATION 9
Expansion of Equation 2.

$$H(z)=\frac{b_0+b_1z^{-1}+b_2z^{-2}+b_3z^{-3}+b_4z^{-4}+b_5z^{-5}+b_6z^{-6}+b_7z^{-7}+b_8z^{-8}+b_9z^{-9}+b_{10}z^{-10}}{1+a_1z^{-1}+a_2z^{-2}+a_3z^{-3}+a_4z^{-4}+a_5z^{-5}+a_6z^{-6}+a_7z^{-7}+a_8z^{-8}+a_9z^{-9}+a_{10}z^{-10}}$$

The actual response will depend on the properties of the noise and the filter characteristics and is most easily determined by simulation.

DIGITAL FILTERS

Digital filtering is a typical application of DSP. Consider a tenth order Chebyshev low pass filter. A variety of sources discuss the development of Chebyshev filters, but I won't go into that here. I used Matlab[3] to develop the following examples. Matlab has a variety of libraries for signal processing and filter design. I used the command `[b,a] = cheby1(10, 0.1, 0.25)` to generate the filter coefficients in Table 1. These are the coefficients of Equation 2, expanded in Equation 9. The ideal frequency response, as calculated by Matlab with double precision floating-point accuracy, is shown in Figure 12.

Matlab can easily calculate the pole zero locations and the magnitude of the pole positions. For a digital filter to be

stable, all of the poles of the characteristic equation must be within the unit circle. The magnitude of the roots of the denominator equation must be less than one. The roots and root magnitudes of the unquantized characteristic equation are shown in the first two columns of Table 2. A plot of the pole zero locations is shown in Figure 13. From the position of the poles, we can see that this is a stable filter when using double precision math.

Consider rounding the pole coefficients to three decimal places, as in Table 3. This still leaves five significant decimal places, about 13 or 14 bits. Figure 14 is a plot of the quantized pole zero locations.

The system is now clearly unstable. Simply rounding the filter coefficients has significantly degraded the filter performance. This situation can be observed mathematically by calculating the roots of the quantized characteristic equation. The roots of the unquantized and quantized systems are

FIGURE 14
Unstable filter poles from coefficient quantization.

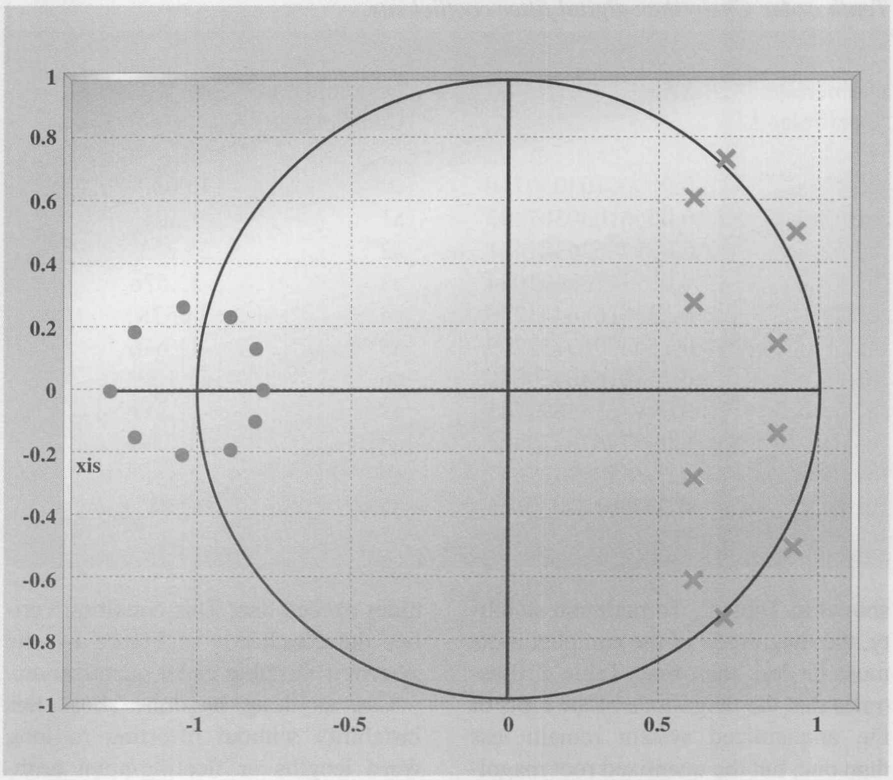


TABLE 3
Tenth order Chebyshev digital filter coefficients.

Numerator Coefficients	x 10^{-3}	Denominator Coefficients	
b0	0.00066103030716	a0	1.000
b1	0.00661030307203	a1	-7.304
b2	0.02974636381481	a2	25.203
b3	0.07932363691054	a3	-53.826
b4	0.13881636441226	a4	78.538
b5	0.16657963752209	a5	-81.626
b6	0.13881636439805	a6	61.109
b7	0.07932363691765	a7	-32.511
b8	0.02974636379882	a8	11.759
b9	0.00661030307647	a9	-2.611
b10	0.00066103030666	a10	0.270

shown in Table 2. To maintain stability, the magnitude of the complex roots must be less than one. Table 2 illustrates that the magnitude of the roots of the unquantized system remain less than one, but the quantized root magnitudes exceed one. This condition verifies the conclusion of Figure 14: the system is unstable under quantization.

Can anything be done about this instability without resorting to long word lengths or floating-point arithmetic? Different implementation forms exhibit different sensitivities to quantization. Rewrite the system in Equation 2 as a product of second order systems, as in Equation 10:

$$H(z)=\frac{(b_{01}+b_{11}z^{-1}+b_{21}z^{-2})}{(1+a_{11}z^{-1}+a_{21}z^{-2})}\frac{(b_{02}+b_{12}z^{-1}+b_{22}z^{-2})}{(1+a_{12}z^{-1}+a_{22}z^{-2})}\cdots\frac{(b_{0k}+b_{1k}z^{-1}+b_{2k}z^{-2})}{(1+a_{1k}z^{-1}+a_{2k}z^{-2})} \qquad (10)$$

In Equation 10, k is equal to the number of second order sections required to implement the filter. There would be an additional first order section for filters of odd powers.

Now consider quantizing the factored coefficients to three decimal places—about 10 bits. The unquantized and quantized roots for the example filter are given in Table 4 along with their magnitudes. Figure 15 is a

A cascade of second and first order sections is going to give the best immunity to the effects of coefficient quantization.

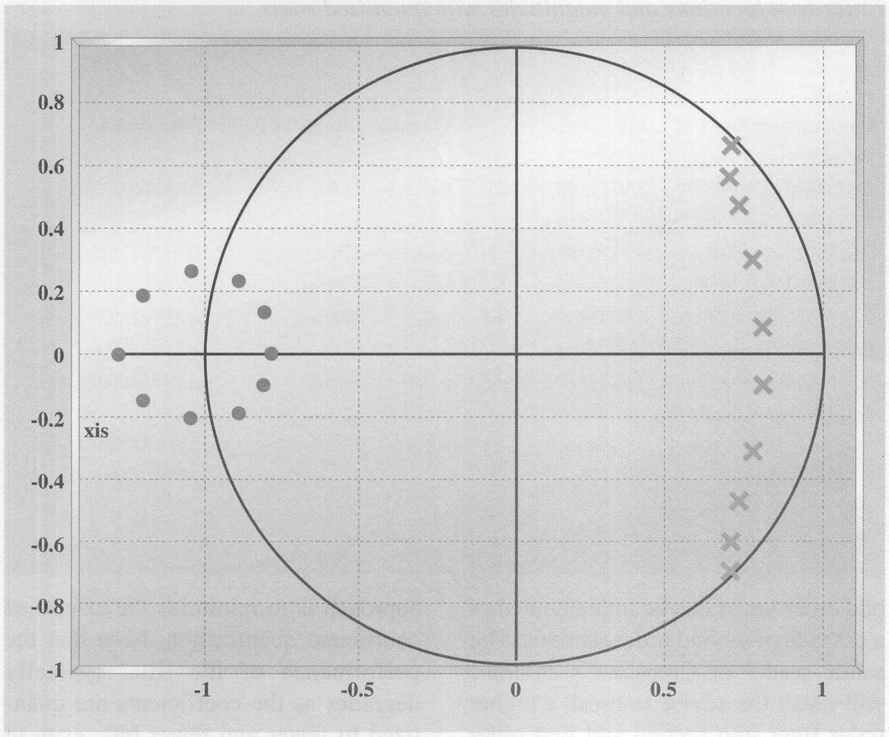
TABLE 4
Filter root locations and magnitudes with quantized roots.

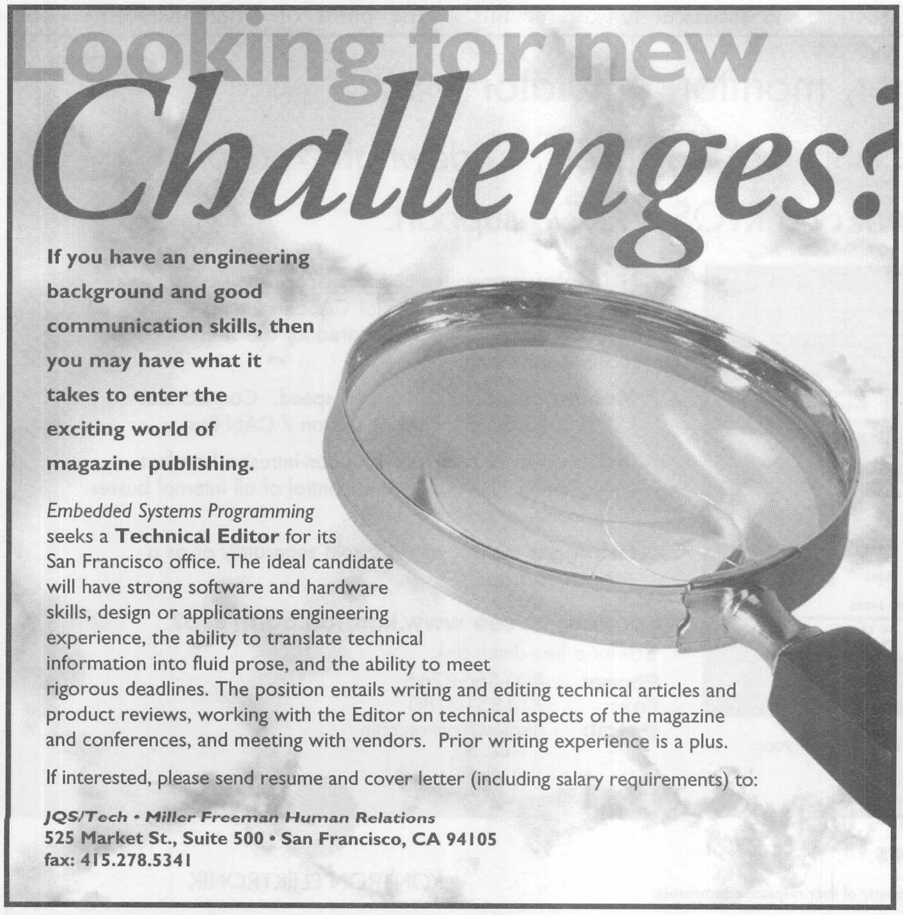
Unquantized Roots of Poles	Magnitude	Quantized Roots of Poles	Magnitude
0.67689787328399 +/- 0.69719981309634i	0.97173983670423	0.677 +/- 0.697i	0.97166763865017
0.68392229911696 +/- 0.61223927931302I	0.91792518560238	0.684 +/- 0.612i	0.91782351244670
0.72193038646088 +/- 0.48422408315146i	0.86928501977167	0.722 +/- 0.484i	0.86921803938943
0.76884440604023 +/- 0.31357503601446i	0.83033187576463	0.769 +/- 0.314i	0.83063650293013
0.80062711808620 +/- 0.10895424215805i	0.80800668877135	0.801 +/- 0.109i	0.80838233528449

plot of the pole zero locations resulting from the quantized roots.

This system is now obviously stable, even when using quantized coefficients. What does this imply for the implementation? The product of second order terms can be implemented as a cascade of second order sections. The astute reader of Crenshaw's columns will recall the advice to break a higher order filter into second and first order sections. A practical reason for this approach is to minimize the effects of coefficient quantization. Note that the performance of the filter typically degrades as the coefficients are quantized to fewer and fewer bits, even to the point of filter instability. This

FIGURE 15
Pole zero map of quantized second order terms.





degradation is true for any of the implementation forms if the word length is too short.

DETERMINING PERFORMANCE

Filters which look identical on paper and are in fact equivalent under continuous math can offer drastically differing performances when implemented under finite precision math. The different forms exhibit different noise susceptibilities and different performance degradation with quantized coefficients. As a rule of thumb, a cascade of second and first order sections is going to give the best immunity to the effects of coefficient quantization. In the end, simulation of the filter is likely to give the most insight into the filter's performance. As I've suggested previously, the use of a numeric manipulation tool such as Matlab is helpful when designing and testing filters. These tools will solve for the filter coefficients for a variety of filters, simulate performance using double precision math, and simulate the performance of quantized filters. The packages typically have a variety of plotting capabilities that enable visualization of filter performance and pole zero locations. **ESP**

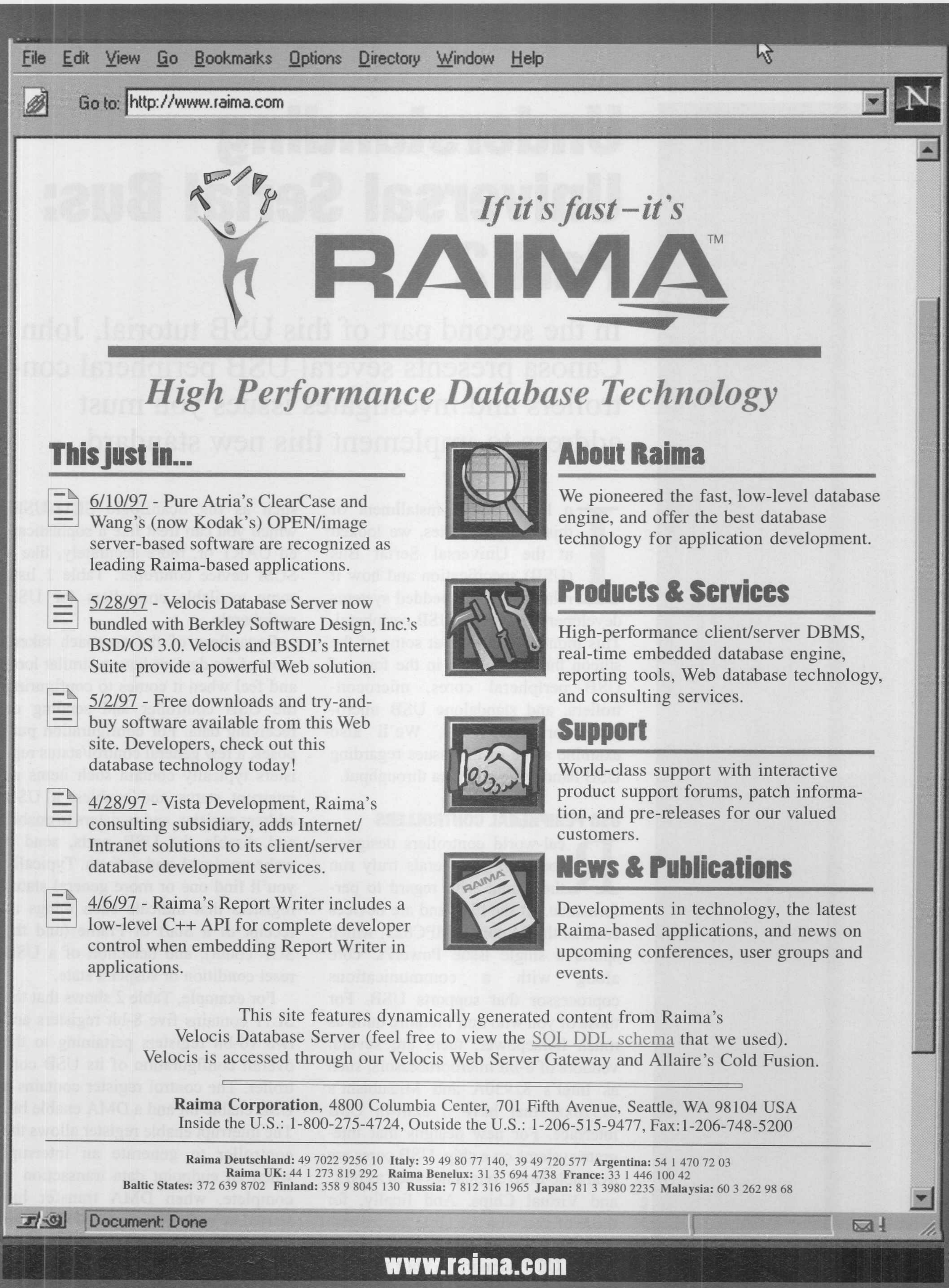
Brad Hunting's industrial experience is in the area of real-time embedded controls and embedded small area networks. He is currently finishing his doctoral degree in mechatronic engineering at Rensselaer Polytechnic Institute. He can be reached at huntib@rpi.edu, or via the Web at cat.rpi.edu/~huntib.

REFERENCES

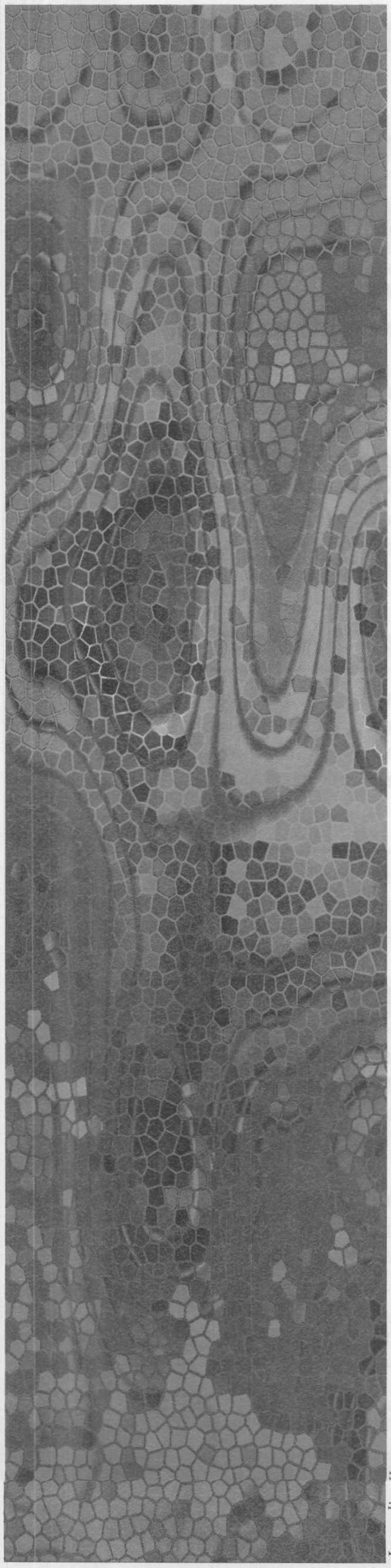
1. Oppenheim, Alan and Ronald Schafer. *Discrete Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

2. Crenshaw, Jack, "Programmer's Toolbox," *Embedded Systems Programming*, May 1996 to November 1996.

3. Matlab, The MathWorks, Inc., Natick, Mass, 01760.


File
Edit
View
Go
Bookmarks
Options
Directory
Window
Help
Go to: http://www.raima.com
N
Document: Done

by JOHN CANOSA



Jacqueline Vansen

Understanding Universal Serial Bus: Part 2

In the second part of this USB tutorial, John Canosa presents several USB peripheral controllers and investigates issues you must address to implement this new standard.

In last month's installment of this two-part series, we looked at the Universal Serial Bus (USB) specification and how it relates to an embedded systems developer designing a USB peripheral. This month we'll look at some of the silicon that's available in the form of USB peripheral cores, microcontrollers, and standalone USB microprocessor peripherals. We'll also examine some of the issues regarding USB bandwidth and data throughput.

USB PERIPHERAL CONTROLLERS

Real-world controllers designed for USB peripherals truly run the gamut with regard to performance. At the high end are devices such as the Motorola MPC823, which sports a single issue PowerPC core along with a communications coprocessor that supports USB. For those of you who don't require quite as much horsepower, there are several vendors of 8-bit microprocessors, such as Intel's 82930A and Mitsubishi's M376xx, that have a built-in USB interface. For new designs that integrate systems on a chip, USB cores are available from companies such as NEC and Virtual Chips. And finally, for those of you who are quite happy with your current microprocessor but want to add USB support to your product, there are standalone USB controllers such as the ScanLogic SL11-USB, which you can treat like a sophisticated UART or, more accurately, like a SCSI device controller. Table 1 lists some available controllers for USB peripherals.

Regardless of the approach taken, most of the devices have a similar look and feel when it comes to configuring the USB controller and sending or receiving data. For configuration purposes, a few general control/status registers typically contain such items as interrupt status and enables, a USB address register, and registers to enable and disable the USB ports, send a wake-up signal, and so forth. Typically you'll find one or more general status registers that indicate such things as receipt of a Start of Frame (and the SOF count), and detection of a USB reset condition or suspend state.

For example, Table 2 shows that the SL11 contains five 8-bit registers and two 16-bit registers pertaining to the overall configuration of its USB controller. The control register contains a USB enable bit and a DMA enable bit. The interrupt enable register allows the controller to generate an interrupt when an endpoint data transaction is complete, when DMA transfer has started or ended, when an SOF packet is received, or when a USB reset occurs. The interrupt status register shows the status of the aforementioned

Real-world controllers designed for USB peripherals truly run the gamut with regard to performance.

interrupt conditions. The USB address register contains the peripheral address that is assigned by the host. A word of caution here: while the SL11 automatically updates this register upon receipt of a `SET_ADDRESS` request, some USB controllers do not, and the application software must update the address.

The final three USB registers in the SL11 are: the current data set register, which contains the status of the data toggle synchronization status for each endpoint; the SOF register, which contains the 11-bit SOF count that is delivered by the host; and the DMA count register, which contains the number of bytes to be transferred via DMA.

The number of endpoints supported by the various controllers range from four to eight. Each endpoint usually has its own configuration register or registers. These registers contain information such as the endpoint number, direction, type (control, isochronous, bulk, or interrupt), data sequence numbers, and the capability to send a stall or a `NAK`. In the ScanLogic SL11 example, five registers are associated with each endpoint, as shown in Table 3.

The endpoint control register contains the expected arm, enable, direction, and similar entries. The base address register points to the memory buffer location for reads and writes. The base length register contains the `MaxPacketSize` information for that particular endpoint. The packet status register contains such information as `ACK` received, transmission errors, timeouts detected, and overflow conditions.

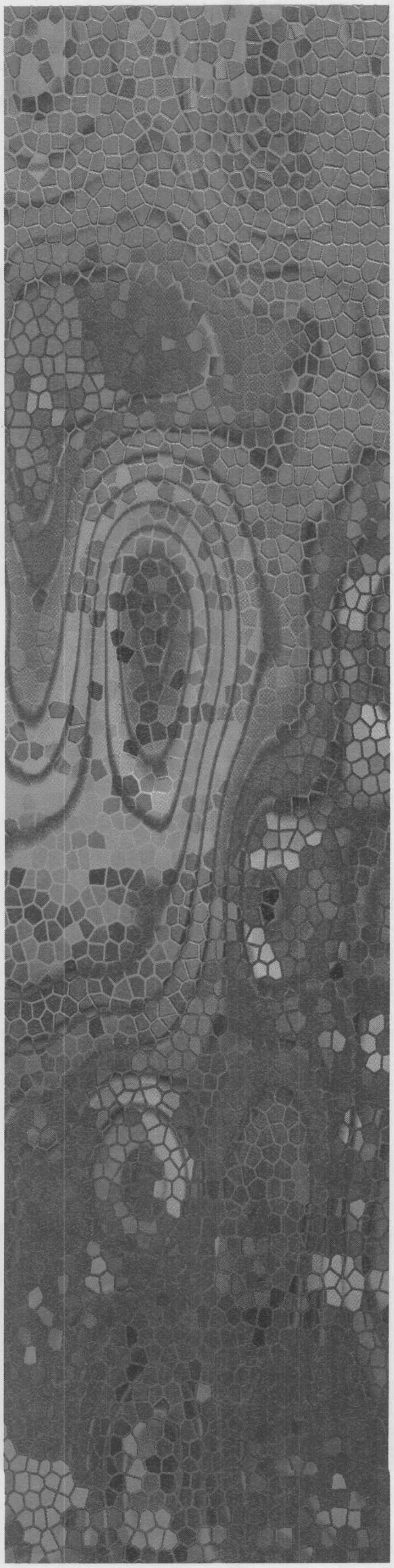
Data movement between memory and the USB is typically accomplished in one of two ways. Some devices, such as the Intel part, use FIFOs to buffer the data, while others allow the user to set up DMA channels to and from endpoints. Intel's 82930A has four transmit FIFOs and four receive FIFOs. Each FIFO is 16 bytes deep, with the exception of the FIFOs corresponding to Endpoint 1, which have a depth of 256 bytes. FIFOs are the typical transfer mechanism for the 8-bit integrated microprocessor/USB controllers and synthesizable cores, which may not have a DMA controller available in the design.

In contrast, both the MPC823 and the SL-11 use DMA to transfer data. As an example, the MPC823 uses its communications processor module (CPM) as the USB interface and the DMA controller. Those who are familiar with the other members of Motorola's MPC8xx family or the MC683xx family will recognize the buffer descriptors (BDs) that are used by the DMA engine to move data between memory and the USB transmit and receive buffers. Don't confuse the term "buffer descriptor" with "descriptor" as defined by the USB specification; it's just an unfortunate conflict of terminology.

Each endpoint has one or more buffer descriptors associated with it. Figure 1 shows the format of a transmit buffer descriptor, which would be associated with an endpoint defined as an `IN` endpoint in the endpoint descriptor. The transmit buffer descriptor consists of a status word, the number of bytes to be transmitted, and a pointer to the region in memory that contains the actual data to be transmitted.

The status word is a bitfield consisting of the following information:

- `R`: Ready
  `0`: Data buffer pointed to by this BD



isn't ready for transmission
1: Data buffer is ready for transmission or is currently being transmitted; the CPM clears this bit when transmission is complete

- W: Wrap
  0: This is not the last BD in the Tx BD table; buffers may be chained so that when the transmission of this buffer is complete, the CPM will automatically move on to the next BD in memory
  1: This is the last BD in the BD table
- I: Interrupt
  0: Do not interrupt after this BD has been serviced
  1: Generate an interrupt after this BD has been serviced
- L: Last
  0: This buffer does not contain the last character of the message
  1: This buffer contains the last character of the message
- TC: Transmit CRC
  0: Only transmit end of packet (EOP) after last data byte
  1: Transmit CRC after the last data byte, then transmit EOP. Unless used for testing, this bit should always be set to one. This field is ignored if the L bit is cleared
- PID: Packet ID
  0X: Do not append PID to the data
  10: Transmit DATA0 PID before sending data
  11: Transmit DATA1 PID before sending data
- TO: Timeout; written by the CPM if the host failed to acknowledge this packet
- UN: Underrun; written by the CPM if an underrun condition is detected during transmission

The typical procedure for setting up a USB transmission by this endpoint would be to prepare the actual data and place it into memory, then write the address of the buffer in the TX_DATA_BUFFER_POINTER, along with the DATA_LENGTH value. Next, all bits of the status register, with the exception of the ready bit should be set to their desired values. Finally, set the ready bit to enable transmission. The next time this particular endpoint is accessed by the host, the data will be transmitted and an interrupt will be generated (if enabled) upon completion of the transmission.

A receive buffer descriptor (Figure 2) contains a status word, the count of the data received (in bytes) and a point-

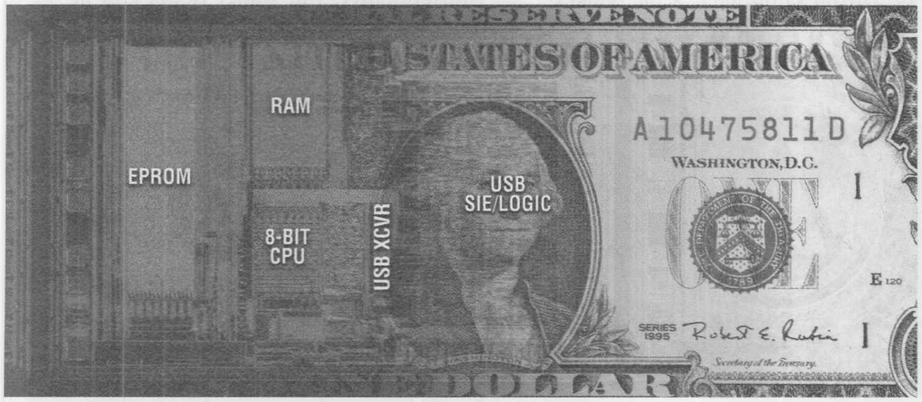
TABLE 1
USB controllers.

Manufacturer	Device
USB Cores	
CMD	USB0676
Future Technology Devices	EMCU USB ASIC
LSI Logic	USB Function Core
Sand Microelectronics	USB Synthesizable Cores
Technical Data Freeway	Synthesizable USB Function Core
Vautomation Inc.	USB Synthesizable Cores
Peripheral Microcontrollers	
8/16-Bit	
Atmel	AT43351
Cypress	CY7C63XX
Intel	82930AX
Mitsubishi	M37690 (Also available as ASIC core)
Motorola	68HC05
NEC	PD789806Y
32-Bit	
Motorola	MPC823
Standalone Microprocessor Peripherals	
Lucent	USS620
NetChip	NET2888
ScanLogic	SL-11
Thesys	TH6503

TABLE 2
SL11 registers.

Register Name	Address (Hex)
Control Register	0x05
Interrupt Enable Register	0x06
USB Address Register	0x07
Interrupt Status Register	0x0D
Current Data Set Register	0x0E
SOF Low Byte Register	0x15
SOF High Byte Register	0x16
DMA Total Count Low Register	0x35
DMA Total Count High Register	0x36

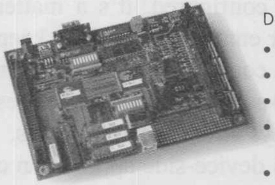
MAKING USB UNIVERSAL: THE FIRST USB µC UNDER $1.00



USB Microcontroller Family

DEVICE	MAX. RAM	MAX. EPROM	NUMBER OF I/Os	PIN/PACKAGE
CY7C630xx	128 Bytes	4 KB	12	20-pin DIP 20-pin SOIC
CY7C631xx	128 Bytes	4 KB	16	24-pin SOIC
CY7C632xx	128 Bytes	4 KB	10	18-pin DIP
CY7C634xx	256 Bytes	8 KB	31	40-pin DIP 48-pin SSOP
CY7C635xx	256 Bytes	8 KB	39	48-pin SSOP

ION and Instant-On Now are trademarks of Cypress Semiconductor Corp.



The easy-to-use ~~$1500~~ now $495! CY3650 USB Developer's Kit includes:

- In-Circuit Emulator board
- Assembler
- Debugger/Monitor
- Software for mouse and joystick applications
- USB code library

http://www.cypress.com/cypress/whathot/hot_top.htm

8-bit USB Microcontrollers for Under $1

Cypress's new family of 8-bit microcontrollers offers the lowest-cost solutions for USB peripherals. These highly integrated USB microcontrollers are built on the industry's smallest RISC core that translates into high volume production with pricing under $1.00.

Full-Feature USB Microcontrollers

Cypress's USB microcontrollers are the first to offer EPROM programmability for fast, easy code customization to speed time-to-market. The USB microcontrollers integrate data RAM, program EPROM, a USB Serial Interface Engine (SIE), and a transceiver for reduced cost and parts count while clock doubler and Instant-On Now (ION™) features ensure low EMI and 70% lower power consumption.

Optimized for a Full Spectrum of USB Applications

Cypress's USB microcontrollers are ideal for all USB peripheral applications, such as mice, joysticks, gamepads, and keyboards. No matter what USB peripheral you're designing, these low-cost microcontrollers make it easy to implement.

Low-Cost Development System for Fast Time-to-Market

Cypress now offers the CY3650 USB Developer's Kit for $495 (a $1500 value). The kit includes a full-speed hardware emulator to help you develop firmware and system drivers for your USB application.

Start Your USB Design Today

Call now to get the USB Literature Kit, including a copy of the USB specification, a Cypress Data Book CD-ROM, and an order form for the USB Developer's Kit.

(800) 858-1810
FAX: 1-408-943-6848
Ask for Kit #T031

CYPRESS

er to the memory buffer containing the data. The status word contains a bitfield with the following structure:

- E: Empty
  0: The data buffer has been filled by the CPM or data reception has been aborted due to an error
  1: The data buffer is empty
- W: Wrap
  0: Not the last BD in the Rx BD table
  1: Last BD in the BD table
- I: Interrupt
  0: No interrupt is generated when this BD is filled
  1: Generate an interrupt when this BD is marked not Empty
- L: Last
  0: Buffer does not contain the last character of the packet
  1: Buffer contains the last character of the packet; either an EOP has been received or an error has been detected
- F: First
  0: Buffer does not contain the first byte of a packet
  1: Buffer contains the first byte of a packet
- PID: Packet ID; valid only if the F bit is set
  00: Buffer contains a DATA0 packet
  01: Buffer contains a DATA1 packet
  10: Buffer contains a SETUP packet
- NO: Error indicating that a number of bits not divisible by eight was received
- AB: Frame aborted; bit stuff error occurred during reception
- CR: CRC error; the received CRC bytes are always written to the receive buffer
- OV: Overrun; a receiver overrun occurred during reception

TABLE 3
Endpoints and addresses.

Endpoint n Register Set	Address
Endpoint n Control Register	xx
Endpoint n Base Address	xx+1
Endpoint n Base Length	xx+2
Endpoint n Packet Status	xx+3
Endpoint n Transfer Count	xx+4

FIGURE 1
MPC823 transmit buffer descriptor.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

OFFSET + 0	R	-	W	I	L	TC	PID	-	-	-	TO	UN	-
OFFSET + 2	DATA LENGTH												
OFFSET + 4	TX DATA BUFFER POINTER												
OFFSET + 6													

FIGURE 2
MPC823 receive buffer descriptor.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

OFFSET + 0	E	-	W	I	L	F	PID	-	NO	AB	CR	OV	-
OFFSET + 2	DATA LENGTH												
OFFSET + 4	TX DATA BUFFER POINTER												
OFFSET + 6													

Setting up an OUT endpoint (which receives data) buffer descriptor involves allocating memory for the buffer and writing the pointer and size into the appropriate BD fields. Subsequently, the status fields can be filled in and the empty bit set. When the empty bit is read as a zero and/or an interrupt has been generated, the receive buffer data can be examined and acted upon.

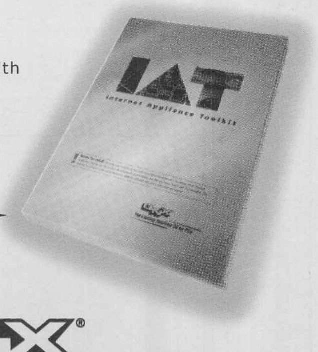
As I've mentioned, the previous examples are representative of a typical USB interface. Some of the terminology of the registers and status bits may differ, but they should have similar functionality. The main differences will be in dealing with DMA or FIFOs. In either case, however, the idea is the same—set up receive and transmit buffers, and move data between them and the USB controller in the proscribed method, either configuring a DMA operation or writing/reading data to/from the endpoint's FIFO.

DEVICE SIDE SOFTWARE

On the surface, design of a USB peripheral's software seems simple. You must have the descriptors defined, stored in memory, and ready for transmission. Once the device is configured, it's a matter of feeding or emptying FIFOs or keeping the DMA controller ready for the next USB transaction. Of course, the design isn't always as simple as it seems.

A USB device-side application consists of four parts: USB communications handling, USB control command parsing and execution, application data transmission and reception, and error handling.

BUILD
THE INTERNET
INTO JUST ABOUT
ANYTHING!
HERE'S PROOF
THIS 1.44M
SELF-BOOTING
WEB DEMO
CONTAINS:
• POSIX-certified RTOS
• Full Windowing System
• HTML 3.2 Browser
• Embedded Web Server
• TCP/IP with PPP
• Internet dialer
QNX
The Leading Realtime OS for PCs
Internet Appliance
Demo Disk—Version


IAT
Internet Appliance Toolkit

USB Basics, Part 2

COMMUNICATIONS AND APPLICATION-SPECIFIC DATA TRANSMISSION

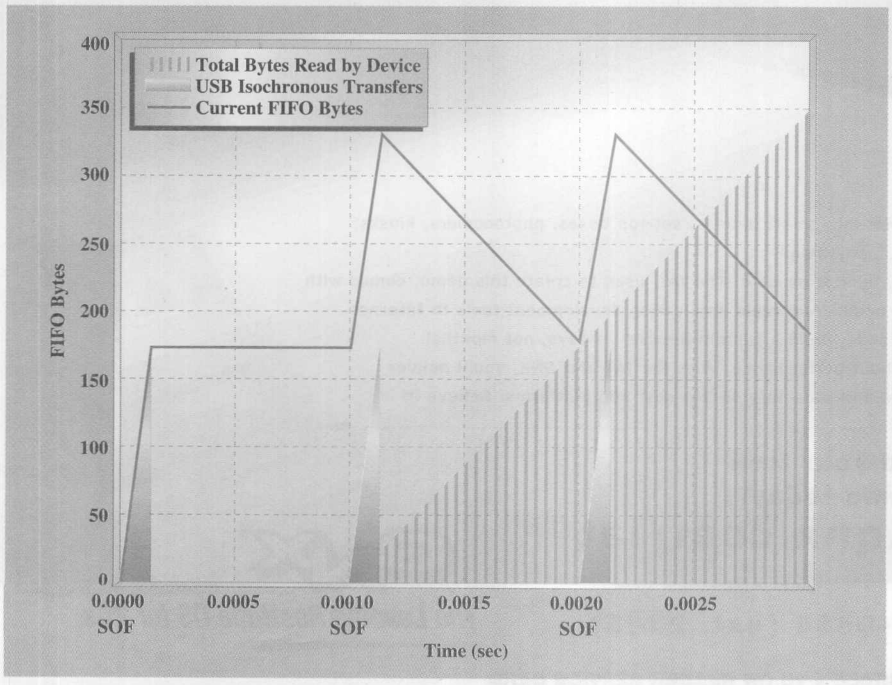
One key area to understand regarding USB is the data buffering system. Buffering requirements are different for each type of transaction.

Interrupt transactions are guaranteed to occur at intervals of 1ms, or an integer multiple thereof, and the amount of data transmitted is limited to 64 bytes. A buffer or FIFO size equal to the amount of one data packet will be sufficient. Once the data is transmitted from the FIFO, the software has a little under a millisecond (at least) to refill it. For most microprocessors, this shouldn't be a problem. As a precaution, the software should program the USB controller to send a `NAK` if the FIFO does not have a complete packet to transmit.

Isochronous endpoints can be a little more complicated. For example, let's say the device in question is a set of stereo speakers that want to drive their D/A converters at a 44.1KHz rate. How big should the input FIFO be? In this case, we have a 1KHz USB frame rate and a 44.1KHz sample rate. The 44.1KHz isn't an integral multiple of 1KHz, so the number of bytes transmitted per frame will vary. We must also consider the buffer latency—isochronous data sinks can't start consuming data received in a frame until the SOF of the next frame is received. From this restriction alone we see that the buffer size must be greater than maximum packet size expected. A quick calculation can give us a rough idea of the buffer size required.

We could transmit 44 samples (at four bytes per sample) per frame (every tenth frame will contain 45 samples to get to 44.1 KHz, but we will ignore that for now). Because we know that we cannot consume any data until the next SOF, after the first frame there will be 44 x 4 = 176 bytes in the FIFO. Once the next SOF arrives, we can start consuming data at our desired rate of 44.1K samples per second, which is equal to 176.4KBps. However, during that time we will also be filling the FIFO with the next USB frame's data at a 1.5MBps rate. This situation is shown in Figure 3.

FIGURE 3
Isochronus buffering.



The isochronous filling of the FIFO can be described with the following equation:

$$\begin{aligned} F &= 1.5E6 * T & 0 < T < 1.7333\mu s \\ &= 0 & T > 1.7333\mu s \end{aligned}$$

where F is the number of bytes added to the FIFO and $T = t - nT$, with t = time (in seconds), T = the USB frame rate (1ms), and $n = 0,1,2,3\ldots$

The number 1.7333μs simply comes from the time it takes to write 176 bytes (44 samples) to the FIFO over the USB. Notice that we are ignoring any packet overhead and assuming that the isochronous transfer starts at the very beginning of the frame, which is the worst case.

At the same time the filling of the FIFO is occurring, we are pulling data from it through the other port. The rate we are emptying the FIFO is simply:

$$\begin{aligned} E &= 0 & t < 1\text{ms} \\ E &= 176{,}400 * (t - 1\text{ms}) & t > 1\text{ms} \end{aligned}$$

Therefore, the total number of bytes in the FIFO is:

$$B = B + F - E$$

where B is the number of bytes in the FIFO. With the FIFO initially empty ($B = 0$), after the first millisecond we can confirm that we have 176 bytes in the FIFO. The maximum number of bytes in the FIFO will occur during the second frame, because the second set of samples is being transmitted over the USB while the device has just started removing data from the first transaction. In this case, the maximum works out to be 332 bytes, a little under twice the packet size. Therefore, the buffer should be at least two times the amount of data that will be transmitted per frame. One word of caution: don't forget that every tenth frame has an extra four bytes associated with it.

The USB specification suggests that bulk transfers normally require a buffer the size of a transmitted packet (64 bytes maximum for bulk transfers).

This is true for devices that are happy with transmitting data only once per frame for a total 64KBps transfer rate. For devices that need to transmit large amounts of error-free data, such as a scanner or digital camera, that transfer rate is too slow. If an application running on the host requests a large amount of data, multiple bulk transactions involving the same endpoint could occur in a single frame. In fact, devices such as scanners rely on multiple transactions during a frame—it's the only way to get close to their desired transfer rates of around 1MBps.

Consider a digital camera that uses 64-byte packets. Table 5-6 of the USB specification indicates the theoretical maximum throughput is 1.216MBps for 64-byte packets.[1] This calculation is theoretical; it assumes no other activity on the bus and that the data is such that no bit stuffing is required—hardly a likely scenario. John Garney's "An Analysis of Throughput Characteristics of Universal Serial Bus" has a more realistic set of calculations regarding USB bandwidth.[2]

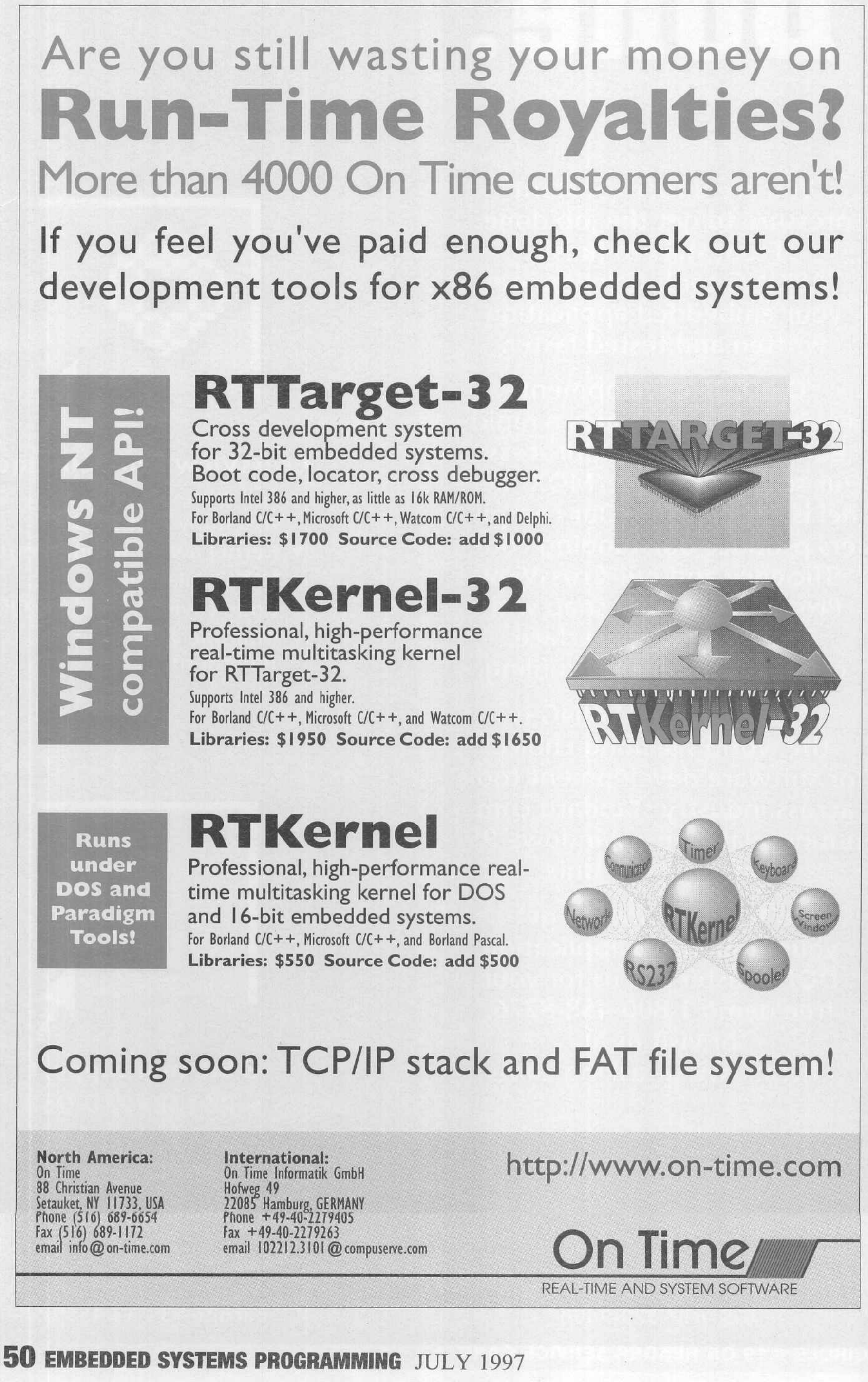
However, our intent is to determine the amount of buffering required on the device side. The key is to determine how much time we have between transactions to fill up a 64-byte FIFO. If we assume back-to-back transactions, the time between the last data byte of the previous transaction being transferred and the requirement for the first data byte of the current transaction to be ready is easily calculated:

```
  16-bit CRC of previous
  2-bit EOP time
  8-bit ACK handshake sync pattern
  8-bit ACK packet
  2-bit EOP time
  8-bit IN token sync pattern
  24-bits IN token packet
  2 EOP times
  8-bit DATA packet sync pattern
+ 8-bit DATA token
  ------------------------------
  88-bit times or 7.3μs
```

(We are ignoring any turnaround and cable delay times)

Note that this calculation assumes an interrupt would be generated immediately after transmission of the last data byte of the previous packet. In reality, the interrupt is most likely generated after receiving the handshake packet, which is the true indication that the transaction is complete.

Just for reference, to get the total transaction time to send an entire 64-byte packet, add 64*8 bit times to the overhead calculated above, and you'll get 50μs. In either case, it's not a lot of time to be handling an interrupt and then filling up a buffer. If the buffer isn't ready for the next transaction, a NAK must be sent. If this is a common occurrence, our throughput can be



severely affected. If we are using 64-byte data packets, sending one NAK between every data transmission will cut our throughput by almost 10%.

When trying to achieve maximum performance, the buffer or FIFO used should obviously be at least twice the size of the data packets. Another solution is to have multiple buffers, so that one may be filled while the other is being transmitted. In fact, having two separate buffers maps nicely into the DATA0 and DATA1 synchronization scheme of USB. One buffer could be defined as the DATA0 buffer and the other would be the DATA1 buffer.

Most controllers allow the user to set up a buffer while another is being transmitted. The ScanLogic SL11, for example, has two buffers. Switching between the buffers is a matter of changing a bit setting in the current data set register. The Intel 82930A is similar in that it lets the user identify two distinct data sets within a single transmit FIFO. The controller will automatically select which data set to transmit, depending on the state of the FIFO. While reading from the proper transmit FIFO data set is automatic, writing to the proper data set is not and must be monitored by the software.

The Motorola MPC823 is a little more sophisticated, allowing the use of many buffers. The software designer could arrange these buffers in a circular queue, with an interrupt generated when each of the buffers is available to be filled. This arrangement could allow the processor to fill up the buffers at a more leisurely rate at the expense of making it more difficult to keep track of which buffer is transmitting DATA0 or DATA1 packets.

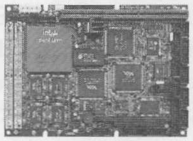
INTERRUPT SERVICE ROUTINES

The preceding discussion on buffering makes it clear that for high performance devices the ISRs of a USB controller must be quick. To reiterate, all USB controller interfaces present a similar software interface, so a discussion of interrupt service routines can look at a specific processor and still be relevant to other devices. In this case, we'll look at the Intel 82930A in a hypothetical scanner application.

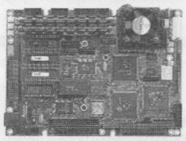
In talking about the ISRs, we need to differentiate between two types. The first type is an interrupt that requires the software to initiate a data transfer, while the second is an interrupt signifying that a packet transmission is complete. If the amount of data required to be transmitted is large, such as a scanned image or a digital camera image file, multiple instances of the second type of interrupt will occur for every one of the first type.

We will assume that the USB controller has been configured and that we will be using Endpoint 1 (which has a 256-byte FIFO) in the bulk transfer

AX10401A



SBC8342



SBC8352

mode with a maximum data size of 64 bytes. We will also assume that the host application has issued a scan command on some other pipe and that it is waiting for data. In this case, the application has registered an IRP with the USB system software, which is sending out IN packets to our bulk endpoint. While our device is scanning and processing the scanned data, we have set the endpoint to reply with NAKs whenever it is contacted. We achieve this by clearing the TX_OE bit in the endpoint control register (EPCON1). To minimize memory usage, our scanned data will be stored in a circular queue of 64-byte buffers. This way we can perform any image processing on the data in chunks, without requiring storage for both the entire scanned image and the processed image.

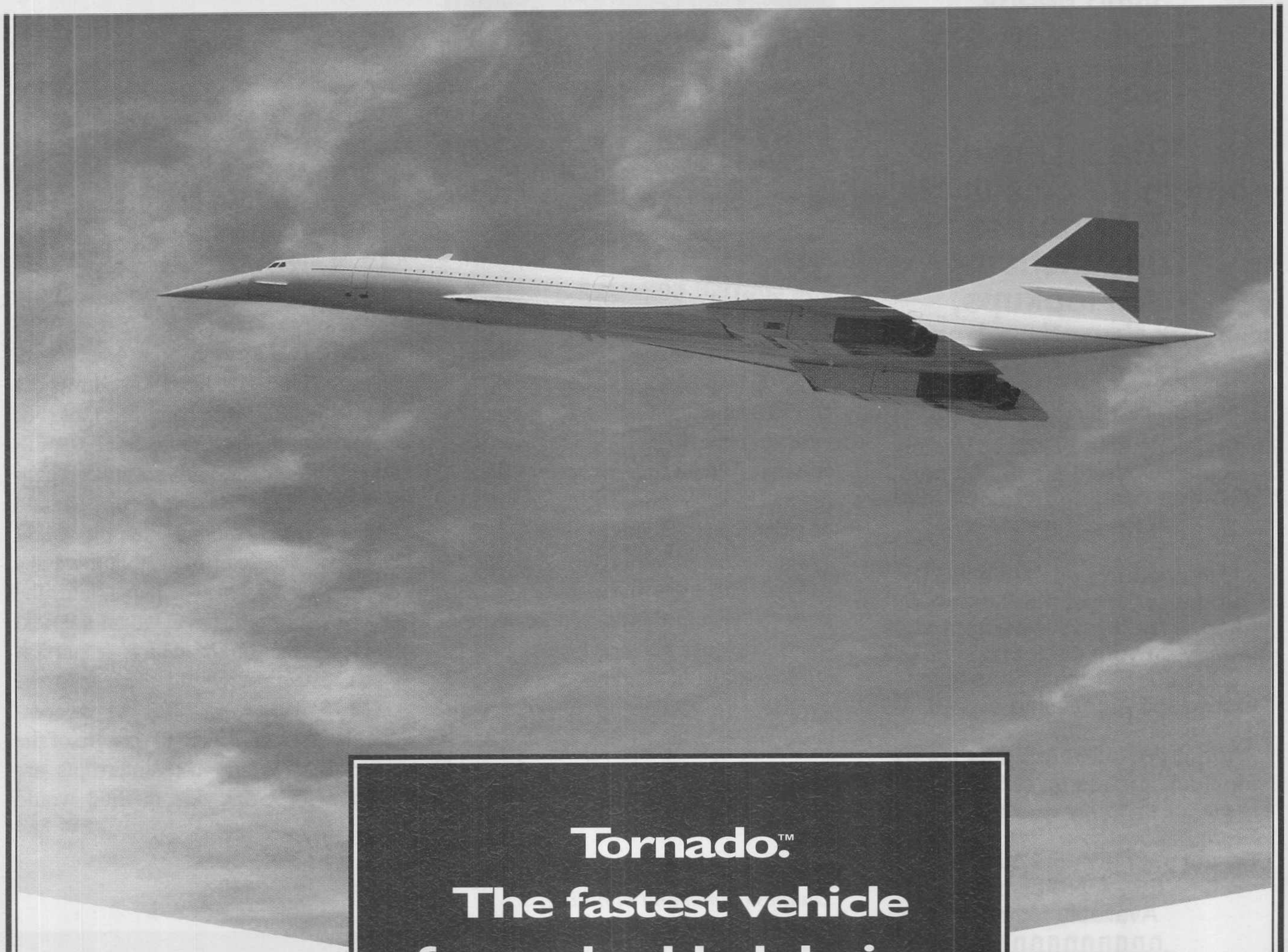
Once the buffer queue is full and ready for transmission, our application generates an interrupt that indicates that fact. The ISR then takes over. The first step is to verify that the FIFO is ready to accept data; if not, we need to notify the calling task and return from the interrupt. If there is room available, we can write to the FIFO and take the following steps:

- Write 64 bytes to the FIFO
- Check the overflow bit (OVF) in the transmit FIFO flag (TXFLG1) register. If it's set, we have an error condition and should set the STL_TX bit in EPCON1. This setting will cause the handshake to a STALL condition and notify the host of a problem. If there was no overflow, we simply write the byte count to the TXCNT1 register
- Next, we should increment our read pointer in the buffer queue. We should not mark the buffer we just wrote as available yet, as there may be an error in the transmission and the data may need to be resent
- Because there is much more data to transmit, we'd like to set up the second data set in the FIFO before we enable transmission. First, check the FIFO index flags to make sure the second data set is available; if it is not, there is an error (we made sure the FIFO was empty before we started). If there is room, we write the next 64 bytes and the bytecount and increment the pointer in our circular queue
- Now that we have both data sets in the FIFO, we can enable the transmission by setting the TX_OE bit in EPCON1 and returning from our ISR

Note that with the 82930A, we have no control over the data sequence number (DATA0, DATA1). The data sequence number is based on the data set from the FIFO that the transmitter is using. Once the serial interface unit (SIU) completes transmission (successfully or unsuccessfully), an interrupt is generated. The ISR for this interrupt should perform the following tasks:

- Because the transmit interrupt can be triggered by any endpoint, we first verify that Endpoint 1 caused the interrupt; if not, we pass on to the appropriate endpoint ISR
- Read and clear the status from TXSTAT1 and clear the interrupt bits
- If the ACK bit is set, the transmission completed without errors. We can advance the FIFO read marker (which starts transmission of the next data set), fill the FIFO with the next buffer from the queue, advance the queue pointer, and mark the buffer whose data we just transmitted as available
- If the ACK is not set, then an error occurred. If the error was not a FIFO underrun (URF = 0), then it was a USB error and we need to retransmit the data. We accomplish this by setting the REV_RP bit in TXCON1. This setting tells the SIU to retransmit the data that is still in the original data set of the FIFO without incrementing the USB sequence number
- If there was a FIFO underrun, there is a serious problem and we should notify the host

Many of the above interrupts will occur during the transmission of the



Tornado.™ The fastest vehicle for embedded design.

Providing the framework for time-to-market advantage.

TORNADO

Getting an embedded design project off the ground is one thing. But it's bringing it in on time that gives you the competitive edge.

That's why we created Tornado,™ the only complete software development environment for embedded systems. So rather than wasting precious months reinventing your software infrastructure for every design, now you can focus on what you do best – innovation.

Tornado includes the industry's most powerful development tools and VxWorks,® our feature-rich, high performance real-time operating system. All seamlessly integrated and a breeze to use.

Tornado is completely customizable. We've published Tornado's API, enabling nearly a hundred Tornado Partners to offer products enhancing Tornado's capabilities and suiting your unique requirements.

And now, we offer Tornado for I2O and IxWorks™ – the first and only development software for I2O, giving engineers a serious head-start in creating software for the new Intelligent I/O standard.

You already know where you're going. Let us get you there faster. Call 1-800-545-WIND or visit www.wrs.com for more information.

WindRiver®
SYSTEMS

CIRCLE # 22 ON READER SERVICE CARD

file, so the key is to keep the buffers ready so that the ISRs will have data to move into the FIFO. Remember, this example is specific to the Intel part, but the general principles will be the same no matter what USB controller is used.

USB CONTROL COMMAND HANDLING

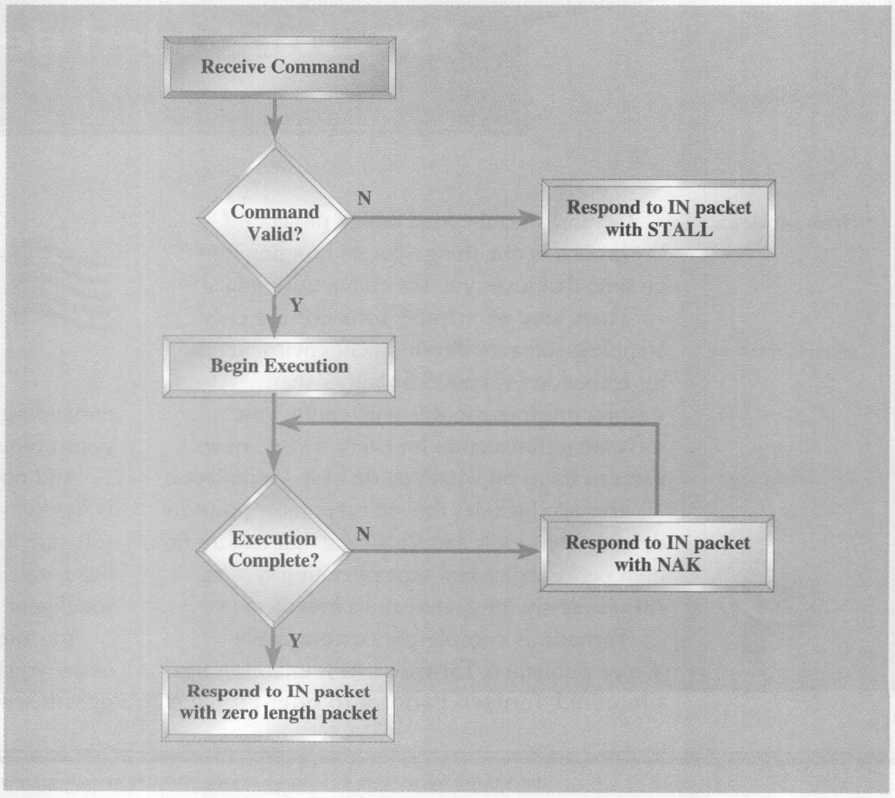
When a device receives a `SETUP` transaction on the control endpoint (Endpoint 0), the application should be aware that an 8-byte command is following it. This command could be a request for data, a command with data associated with it, or a dataless command.

It's the responsibility of the device software to parse the command, perform the required task, and generate a status response as shown in Figure 4. Of course, if the command is one that required the device to respond with data, such as a `GET_DESCRIPTOR` command, the flowchart in Figure 4 would not end with a zero length data packet, but with a packet containing at least a portion of the requested data.

ERROR HANDLING

Not to sound paranoid, but potential sources of errors lurk everywhere. From noisy electronics to unsupported commands to application errors, a USB device should be able to handle errors with aplomb. The `STALL` handshake turns out to be a useful error handling tool. For example, if the host sends a `SET_CONFIGURATION` command with an invalid configuration number, the device should return a `STALL`. Another example would be if a device determines that there is a device error such as a paper jam, the device could return a `STALL` on the next USB transaction in which it participates. The host would then send the `CLEAR_FEATURE - ENDPOINT STALL` command and inquire as to the nature of the error. Because no true interrupts are defined for USB, this method would

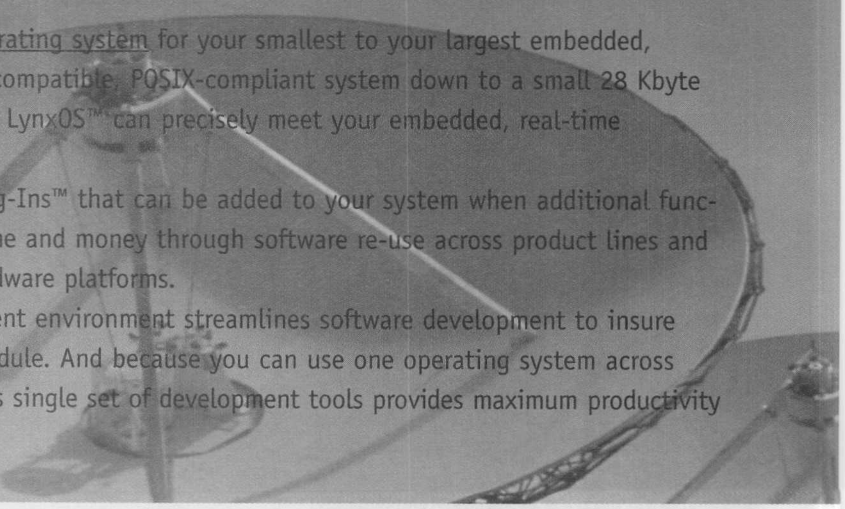
FIGURE 4
Dataless command handling.


Receive Command
Command Valid?
N
Respond to IN packet with STALL
Y
Begin Execution
Execution Complete?
N
Respond to IN packet with NAK
Y
Respond to IN packet with zero length packet

LynxOS (actual size)

Only LynxOS is fully scalable across all of your communications products.

LynxOS (actual size)



Now you can standardize on a single operating system for your smallest to your largest embedded, real-time applications. From a full featured Unix-compatible, POSIX-compliant system down to a small 28 Kbyte microkernel – and everything in between, flexible LynxOS™ can precisely meet your embedded, real-time performance and deterministic requirements.

LynxOS employs Kernel Plug-Ins™ that can be added to your system when additional functionality is required. You save time and money through software re-use across product lines and portability across all popular hardware platforms.

Our PosixWorks® development environment streamlines software development to insure your product is delivered on schedule. And because you can use one operating system across your entire range of products, this single set of development tools provides maximum productivity for your engineering teams.

LEARN MORE ABOUT LYNXOS IN OUR FREE REAL-TIME NEWSLETTER.

(Not actual size)

Call 1-800-255-LYNX
www.lynx.com

Committed to your success in real-time.

Lynx
REAL-TIME SYSTEMS, INC.

CIRCLE # 24 ON READER SERVICE CARD

provide the host with the most immediate notification of an error.

FLEXIBLY COMPLEX

Those who have done the null modem/gender changer/ 9-pin-to-25-pin adapter/RS-232 dance will appreciate what USB will do for serial communications. The plug-and-play aspects of USB make it extremely attractive to the average consumer and a godsend to those who have struggled through the alternatives.

The types of devices that will use USB include telephones, modems, keyboards, mice, 4x and 6x CD-ROM drives, joysticks, tape and floppy drives, scanners, digital cameras, and printers. USB's 12Mbps data rate will also accommodate a whole new generation of peripherals, including MPEG-2 video-based products, data gloves, and digitizers. Because computer/telephony/consumer integration is expected to be a big growth area for PCs, you can expect to see Integrated Services Digital Network (ISDN), ADSL, and digital PBXs, as well as a generation of products yet to be thought of.

As with most things electronic, with flexibility comes complexity, and USB was designed to be a very flexible bus. The software aspects of USB should not be underestimated, both on the device and host sides. On the device side, cost and time-to-market pressures combine to create some very software-intensive designs. These pressures, along with the USB specification, also place some demanding requirements on the device software, including code size, speed, and buffer size. Only a thorough knowledge of USB will allow a designer to deliver a product that meets all of those requirements.

While initial forecasts predicted that by early 1997 USB would be standard for all PCs shipped, that prediction has slipped a bit. As I mentioned before, Microsoft has been a bit late in releasing its new driver model and the associated class drivers and minidrivers. The current estimates are that USB will be ubiquitous in 1998. This slippage has merely prolonged the inevitable, but it does reinforce what most of us involved in embedded systems design have known for a while—software is critical. ESP

John Canosa is a principal member of the technical staff at Questra Consulting, where he is responsible for designing and developing hardware and software for embedded designs.

REFERENCES

1. *Universal Serial Bus Specification, Rev. 1.0.* Hillsboro, OR: USBIF, 1996.

2. www.intel.com/embedded: Garney, John, "An Analysis of Throughput Characteristics of Universal Serial Bus," Media and Interconnect Technology, Intel Architecture Labs.

OTHER SOURCES

Anderson, Don. *USB System Architecture.* Reading, MA: Addison-Wesley, 1996.

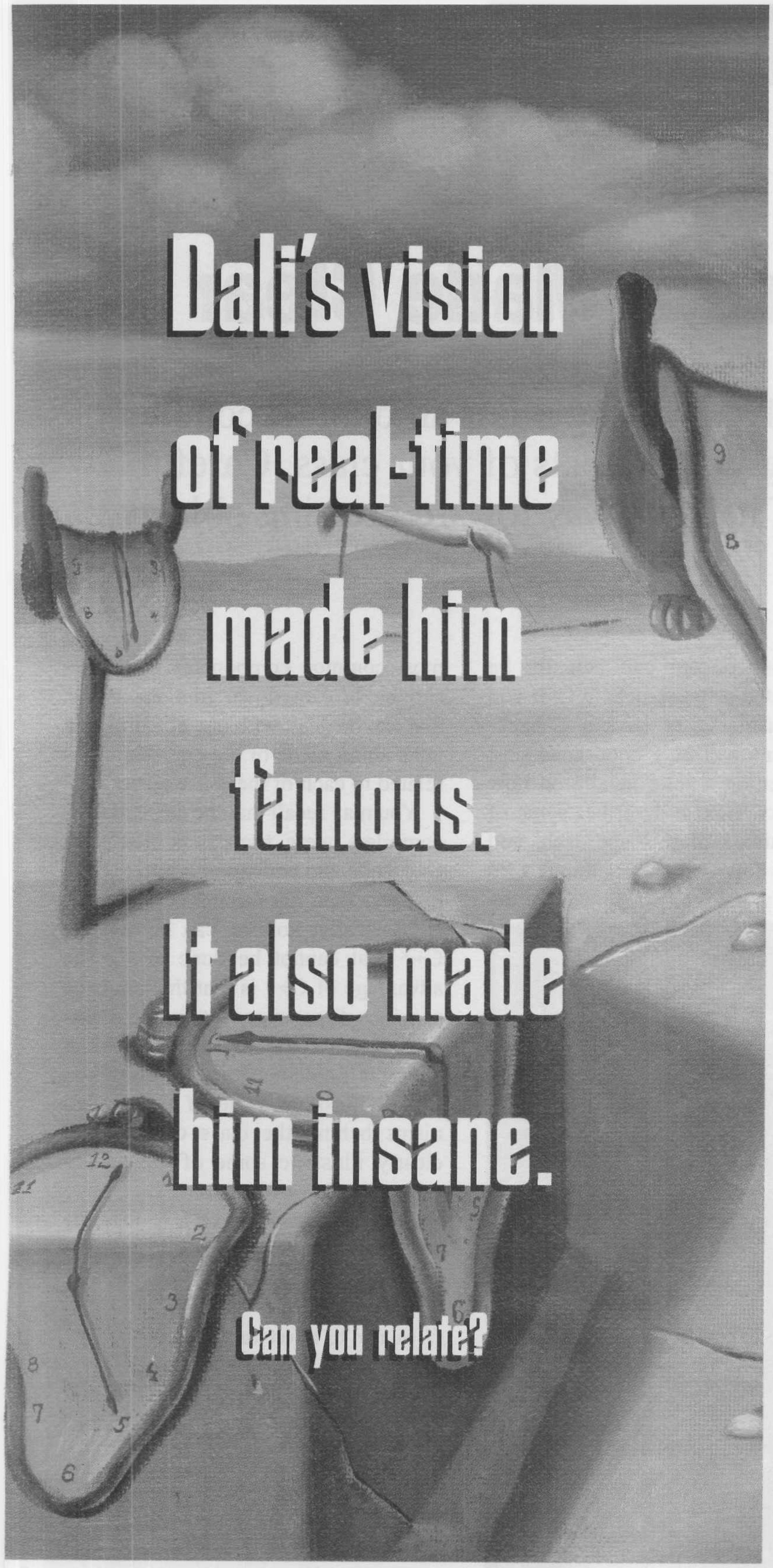
Intel 82930A Universal Serial Bus Microcontroller User's Manual. Santa Clara, CA: Intel Corp., 1996.

Motorola MPC823 Specification. Austin, TX: Motorola Semiconductor Products Sector, 1997 (http://www.mot.com/SPS/ADC/pps/prod/eppc/mpc823.html).

ScanLogic SL11 USB Controller Technical Reference. Bedford, MA: ScanLogic Corporation.

www.mindshare.com: USB Training Courses, Mindshare Inc. (719) 488-8990.

www.usb.org: *The USB Implementer's Forum Home Page.* Source of information about the USB, including the specification and pointers to USB products, controllers, cores, and so on.


Dali's vision
of real-time
made him
famous.
It also made
him insane.
Can you relate?

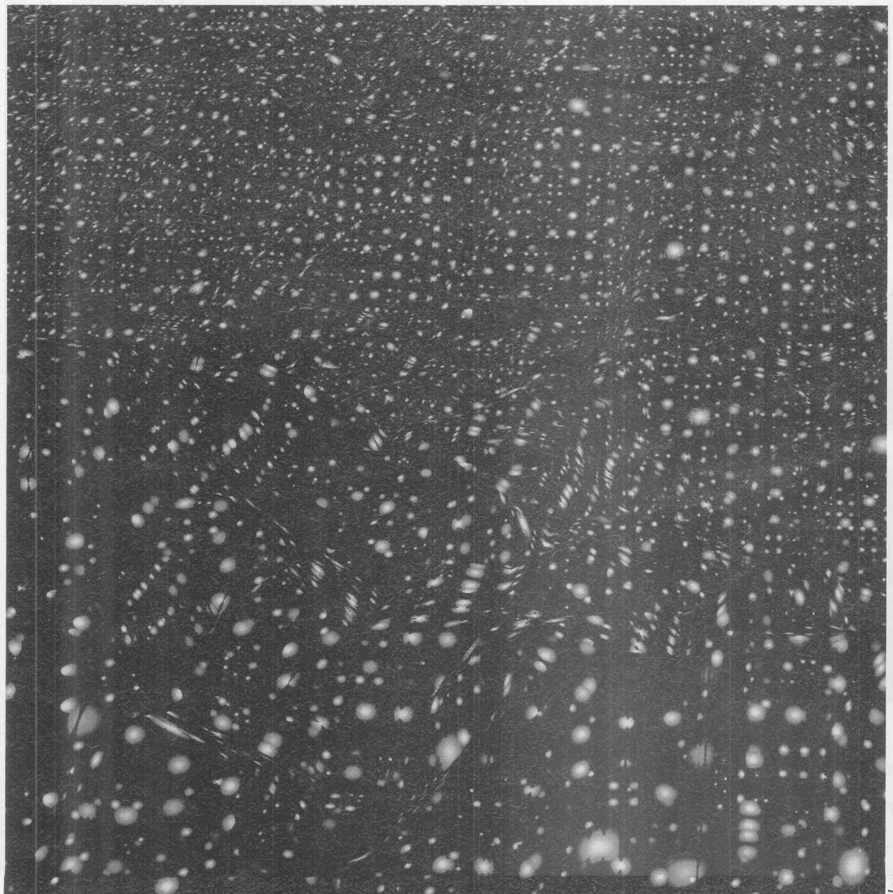
Overcoming
the Crisis in
Real-Time
Software
Development

by RANDALL A. MADDOX

The Class Concept Made Clear

Developing C++ classes involves writing a fair amount of code to assist the compiler in working with objects of your class. Careful attention to detail can pay off in a big way toward creating objects that are useful abstractions.

Writing classes in C++ isn't always an easy task, but it can be made less stressful. This article presents an approach that can take a lot of the guesswork out of the process. It starts with an example showing how many of the mechanisms behind the C++ class concept can actually be implemented using `structs` in C. If you are still using C, or have a C background, this will be a good head start toward getting a solid handle on how C++ works. Next it describes some of the limitations of this approach, and illustrates how C++ makes things a lot easier for you. Finally it shows how the most common components of classes can be broken down into categories that can be dealt with one at a time and talks about some of the considerations related to each of these categories.



Rupert Adley

You may recall that the original C++ compilers were translators that took C++ code and converted it into C code that was then run through a C compiler. Basically, nothing can be done in C++ that cannot be done in C. The advantage of C++ is that the language provides direct support to make it easier and more natural. Taking a look at a simple implementation of classes in C will illustrate some of the mechanisms behind the class concept and clearly illustrate some of the advantages that C++ offers over C.

Imagine that we have a header file with the following `typedef` and prototypes:

```
// pointer to object data
typedef struct This * const thisPtr;

// constructor
thisPtr Create();

// destructor
void Destroy(thisPtr tp);

// member function
int F(thisPtr tp, T parm);

// another member function
int G(thisPtr tp, T parm);
```

The `typedef` for `thisPtr` is called an *incomplete type specification*. The

The advantage of C++ is that the language provides direct support to make it easier and more natural than C.

typedef says only that a thisPtr is a constant pointer to a structure that is not yet defined. This definintion is sufficient to allow variables of type thisPtr to be declared and used as function parameters and return values, but the operations allowed on a thisPtr are restricted to only those for which no information about the structure that is pointed to is required. That is, code that doesn't have any more information about the This structure than what is provided in the header file cannot perform any operation that involves the (unknown) structure members. Neither can it perform an operation that requires knowledge of the size of a This structure or dereferencing of a thisPtr. All information about what a thisPtr points to is still hidden. In Modula-2, this would be called an *opaque* type because external users of the type can't see into it.

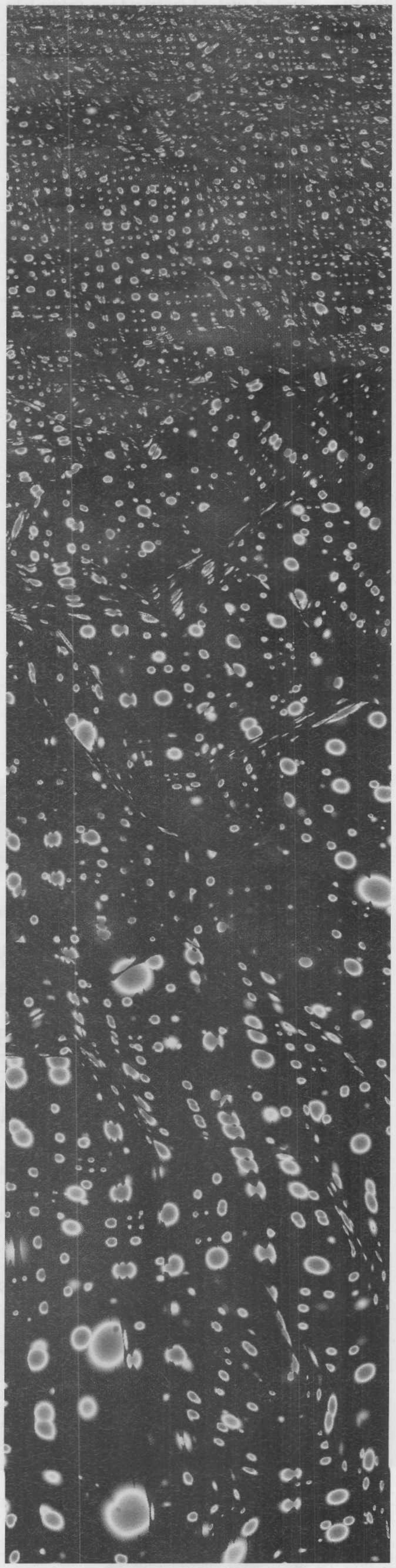
In the corresponding implementation file (.cpp, .cxx), the compiler will need to see a fully elaborated typedef of the This structure so that the functions defined in that file can work with the internals of the structure:

```
typedef struct
{
   T1  dataElem1;
   T2  dataElem2;
   .
} This;
```

Now we can provide an implementation of the Create() and Destroy() functions, as well as any other "member" functions we choose to provide:

```
thisPtr Create()
{
   //allocate memory for internal data
     representation
   thisPtr tp = (thisPtr)
     malloc(sizeof(This));
   // if allocation succeeded,
   // initialize the structure
   if(tp)

     tp->dataElem1 = xxx;
     tp->dataElem2 = xxx;
     .
   }
   return tp;
}

void Destroy(thisPtr tp)
{
   if(tp)
   {
     // do any necessary cleanup operations

     // free the memory allocated
     // by Create()
     free(tp);
   }
}
```

There are several points of interest to note here. Although external code that includes our header file can declare variables of type thisPtr, they can only assign a value at the point of declaration, because the typedef specifies that a thisPtr is a constant pointer. Once declared and initialized, a thisPtr cannot be modified (at least not without violating the rules of fair play by casting away the const-ness). Outside our implementation file, where the elaborated typedef of the This structure is not visible, the expression sizeof(This) is a compiler error. Thus, the only way that external code can

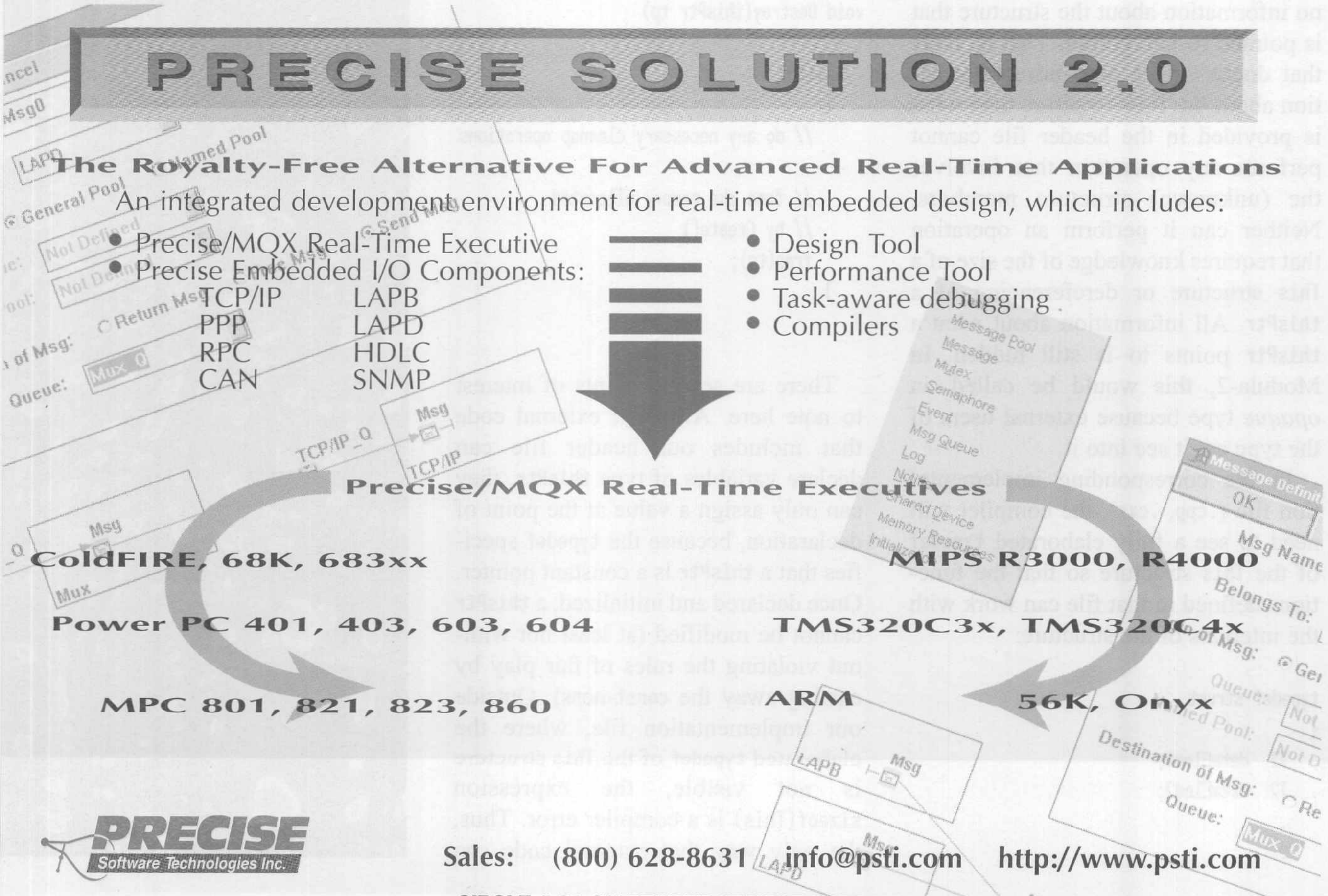


assign a valid initial value to a variable of type `thisPtr` is by calling our `Create()` function, which not only provides a valid pointer value, but also guarantees that the pointed-to structure has been properly initialized. Each one of the member functions must take a `thisPtr` as its first parameter. This is the distinguishing characteristic that makes them member functions. Only our member functions have knowledge of the internal details of the `This` structure. Thus, only our member functions can manipulate the data elements of a `This` structure.

So what have we accomplished with this exercise? We have effectively created a new type that external code can make use of. This new type is opaque in the sense that these external users cannot know anything about the internal structure of the type, and the operations available are only those that we choose to provide as member functions. Only our member functions can manipulate the internal structure; that is, we have achieved data hiding and encapsulation. Valid initial values for variables of the new type can only be obtained from our `Create()` function, and those are guaranteed to be properly initialized. Any operations that result in the internal structure containing invalid values can only occur in one of our member functions—in other words, we don't need to worry about external code messing with our internal structures. And our `Destroy()` function will guarantee that proper cleanup is performed as needed.

While this simple example does demonstrate many of the benefits of the class concept, there are holes, of course. A variable of type `thisPtr` can be declared and initialized with a non-valid value, and the compiler won't complain. The compiler cannot guarantee that every structure allocated and initialized by our `Create()` function will be destroyed via our `Destroy()` function, thus leaving open an easy path to memory leaks. And the compiler cannot assure us that our member functions will never be called with a `NULL` or an otherwise invalid `thisPtr`. We either have to trust our external users or check for this situation in every member function.

All of these weaknesses can be summed up by saying that we are relying on programmer discipline to ensure correct usage. Compiler assistance is limited. In C++, on the other hand, the compiler guarantees that the class constructor and destructor will be called as necessary and that the class member functions will always be passed a valid `this` pointer. C++ also allows us to structure our types hierarchically, so that types further down the inheritance tree can reuse code and behavior from their parent classes. This structure also ensures that only typesafe conversions amongst different types are allowed. That is, C++ classes are directly bound up with the compiler type checking system, while our simple example doesn't provide this level of support.



hitex DEVELOPMENT TOOLS

Genuine Real-Time Emulators!

MORE Powerful New Products at Hitex Emulators that work!

8051 AX51 & MX51 Emulators

Hitex is Europe's leading 8051 emulator supplier. Well over 150 derivatives from manufacturers such as Intel, Dallas, Philips, Siemens, Atmel and AMD are supported. Recent additions include the Siemens 515C (CAN interface) and 504L series and the Intel 87C51FC... with more to come soon. Hitex probes are the smallest in the industry and each probe can support many variants lowering your costs and maximizing the utility of your Hitex equipment. Hitex has both bondout and standard chip Emulators offering an unbiased selection to you. Hitex 8051 emulators are powerful with features that speed your projects to completion. For a list of these Hitex Competitive Advantages, see our Web site or phone us today. Hitex Hardware Breakpoints stop the emulation before executing the instruction where the breakpoint is set. A Complex Trigger system coupled with a filterable Trace makes this a most powerful emulator.

MCS®251 AX251 Emulator

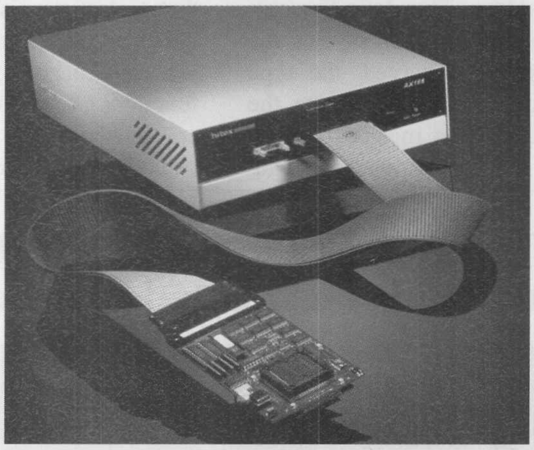
The AX251 is the latest 16 bit emulator design from Hitex for the Intel MCS®251 microcontroller family. Trace memory and all software is included in the purchase price. The Intel 8x251SB microcontroller is supported and future variants will require only a pod change or a software driver. The Universal Serial Bus (USB) chips (8x930AX & 8x930HX) are also supported increasing the versatility of the AX251. The AX251 operates in genuine Real-Time and does not require any resources from the target system which facilitates effective debugging sessions for the designer. The advanced AX251 is shipping from stock today and technical support is based in California. Hitex products are sold world-wide.
The hardware breakpoint system consists of separate 256K execution and 256K data access breakpoints. You can set a breakpoint on a variable name and an address. Breakpoints stop the emulation before executing the instruction where the breakpoint is set.

Siemens C161 AX161

Siemens introduces a new low cost processor - the C161 - Hitex is now pleased to announce the new low cost AX161 dedicated C161 In-Circuit Emulator with a full-featured bundled Keil 161 C Compiler. This emulator is shipping now - when needed. It utilizes Hitex standard chip emulation techniques eliminating version problems. The AX161 is functionally equivalent to the popular AX166 emulators. At a price low enough to please everyone ....
It has 20 hardware breakpoints, 2 Complex Trigger systems, 4 K frames or 2048 C Lines trace buffer, 256 Kbytes 0 WS memory and a Real Timer. Genuine Real-Time operation - no cycle stealing. Emulator functions can be accessed while the emulation is running without stealing target cycles. The AX161 uses same friendly HiTOP user-interface as all other Hitex products. This unit will have upgrade capabilities to Bondout technology and other derivatives.

Siemens C167 AX166

Hitex offers the widest range of emulators available for the entire Siemens 166 family including the 166, 167, 167CR, 165, the 163 and now the just announced 161. Both Bond-out and regular series chip emulation are offered to provide you with the emulator you really need. These are supported by the full feature teletest 32 and the economical AX series. The AX is now available in a bond-out version and is shipping NOW! The Hitex Bondout emulators can trigger on internal register values in Real-Time! This is an exclusive Hitex feature. Various derivatives are supported by changing the pod, a personality module or merely the chip on the pod. Your Hitex emulator will not become obsolete with the introduction of new derivatives. These emulators operate in genuine real-time at full processor speeds for powerful emulation. Clock cycles are never stolen due to the distributed processing in the emulator.



AX166/161 In-Circuit Emulator - Standard Chip

CAN Network CANalyzer

Hitex distributes the Vector CANalyzer - the world's standard for CAN (Controller Area Network) simulation, testing and debugging equipment. The CAN bus is becoming popular for automotive and factory automation applications in North America. The CANalyzer can generate messages, gather statistical network information, simulate virtual nodes and reply to specified network messages. A intuitive user-interface provides a powerful point & click environment for fast setup and use of the analyzer. It supports both 11 and 29 bit identifiers. The CANalyzer comes in a ISA bus card and also a PCMCIA card for laptops. The software is Windows or DOS. Hitex also offers a 8 bit ISA card for user applications. The **HiCAN** card is accessed using standard IO space and contains an on-board processor to ensure traffic is handled without loss or corruption of messages. See our Web site or free CD-ROM for general CAN info.

Hitex Development Tools
2055 Gateway Place #400
San Jose, CA 95110

☎ (800) - 454-4839
☎ (408) - 298-9077
Fax: (408) 441-9486
email:info@hitex.com
http://www.hitex.com

Hitex also supports:

8051	68HC11
683xx	386EX/DX
186EA/EB/EC/EM/ES/ER	

CIRCLE # 27 ON READER SERVICE CARD

The concepts in this example do, however, illustrate very well how C++ actually works. Each object of a class does have memory allocated for the object's hidden internal data representation. The purpose of the class constructor is to provide guaranteed initialization of that memory. The purpose of the destructor is to provide any necessary cleanup before that memory is deallocated. A pointer to the internal data representation is correctly passed to all class member functions. And only those member functions have knowledge of the internal data representation of the class.

CLASSES IN C++

So what is a class in C++? First and foremost, a class is a user-defined type. This definition carries with it many implications. For one thing, as already observed, the type hierarchy directly corresponds to the class hierarchy. This means that you can use the compiler's type checking system to enforce many of your class design decisions. You can also use the compiler to enforce other design decisions related to scope and visibility. We won't go into this concept any further here, but keep in mind that the way you design a class, and many of the functions you write for a class, will help the compiler to enforce your design decisions. The more problems you can catch at compile time, the fewer problems are left to catch at run time. Fixing compiler errors is always cheaper, easier, and faster (to say nothing of less embarrassing) than tracking down some mysterious run-time bug that causes your software to crash in the middle of the night while processing your customer's most critical data.

At a more practical level, we can say that a class consists of data and functions that operate on that data. In other words, a class consists of data and functions (behaviors) that "go together" in the sense that the data and behaviors together provide a coherent model of something. I deliberately introduce the term "behaviors" here because the functions provided by your class implement the behaviors the class exhibits.

For example, if you have a class `Tree`, then some data that might make sense would include the species and age of that `Tree`. Some appropriate behaviors might include `DropLeaves()` for the fall and `GrowLeaves()` for the spring. The point here is that before you can begin to design a class, you must have a clear picture of what the class is and how it should be expected to behave.

Classes don't just spring forth out of the ether—they are deliberately designed to provide a software model of something that a program needs. If you cannot identify a specific need for the class in your program—some necessary role for objects of the class to play—then you are wasting time even thinking about it. If you cannot identify a single clear concept or thing that

still think Microtec is just a tools company.

But we're so much more. Our complete line of products and services can differentiate your designs, add value, and get you to market faster.

Products like Spectra®, Microtec's fully integrated—yet remarkably open—embedded development environment. It includes VRTX®, the industry's most powerful and proven RTOS, as well as our renowned XRAY® debugger and C/C++ compilers. Use them together, separately, or with your own choice of RTOS and tools.

Or let us do it for you. Microtec offers a full range of consulting and professional services, so we can provide everything from off-the-shelf point products to completely integrated, turnkey solutions.

So if you're ready to slice serious time and money off your next development project, call Microtec at **1-800-950-5554, Dept. 300**. Mention this ad, and we'll send you your own copy of the Embedded Software Expert badge.

Think of it as a little reminder from your total solutions company.

Microtec®
A Mentor Graphics Company

www.mri.com

CIRCLE # 29 ON READER SERVICE CARD

UL REGISTERED FIRM

Class Concept

the class is intended to model, then you aren't ready to begin designing, much less implementing, a class. You can only design a class after you know what it is and why you need it.

Figure 1 illustrates the common components of most classes. The following paragraphs present a discussion of each of these components and some of the considerations related to their design and implementation.

DATA

The data associated with a class is often referred to as the *attributes* of the class. The set of values of those attributes at any given point in time constitutes the *state* of the class. Because the data is initialized by the constructor to a valid state, and the data is thereafter manipulated only by member functions of the class, then it is clearly the responsibility of the class to keep its data in a valid state. It is extremely desirable that any invalid state of an object should be traceable to code within the object's class. Maintaining this traceability as an invariant ensures that the first stage of debugging—the localization of the source of the problem—is already accomplished when the code is written, before it has ever been executed.

Class data may be specified to be per class or per object. Per class data, denoted as *static*, is shared between all objects of the class; that is, static class data is associated with the class and not with any individual object. This association can be useful for collecting class-wide statistics or for communication between objects of a class. Per object data, a separate instance of which is allocated for each object of the class, is generally more common than static data.

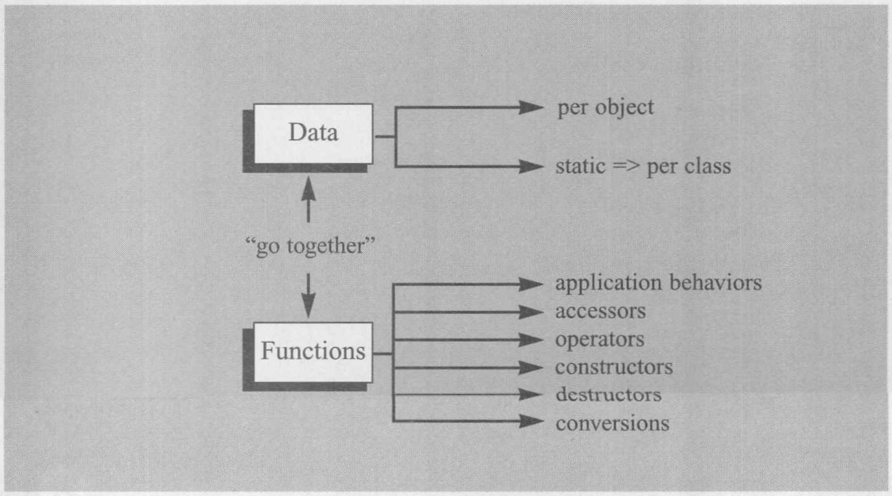
PER OBJECT DATA

Storage for per object (*non-static*) data members is allocated when an object comes into existence. The constructor is called to initialize this storage to a known good state. Per object data is what the object's `this` pointer points to. This situation parallels our earlier simple example of classes in C.

PER CLASS (STATIC) DATA

Storage for per class (static) data members is allocated when the definition of that data is encountered by the compiler or when the implementation (`.cpp`, `.cxx`) file is compiled. Thus, to ensure that class static data starts out in a known good state, it is required that this data be initialized at the point of definition. If no initial values are provided, then this storage will be initialized at program startup,

FIGURE 1
Class components.



THEM
US

before `main()` is entered, to contain all zeroes, as per the ANSI C Standard. This condition may or may not be acceptable.

FUNCTIONS

Many functions may be associated with any class. These functions can be conveniently classified into logical groups corresponding to where the requirement for the function originated.

Some functions are necessary to provide the application-specific behavior that objects of the class will exhibit. Others provide controlled access, if necessary, to class data members. These are the functions that must be considered at design time. These are also the functions that users of your class will probably be most familiar with, because they provide the behaviors and access needed by other parts of the program.

A fair portion of the code in most classes consists of functions that control how objects of the class come into existence, how they are destroyed when their existence ends, what operators may be applied to objects of the class, and what types objects of the class may be converted to and from. These are the functions that must be considered at implementation time. Many of these functions are implicitly invoked by the compiler, and may not ever be explicitly invoked by a user of your class.

The main benefit to be gained from separating class functions into groups this way is that you can think about each group separately, rather than considering the whole function at once. While deciding what application-specific behaviors your class must support, you do not need to also be thinking about how the class destructor will work, or what conversion functions you should supply—divide and conquer. Get those class functions into groups small enough so that you can focus on related concerns without distractions. This will lead to better designs and implementations.

APPLICATION BEHAVIORS

As noted, these functions implement the class behaviors required by the program, usually based on modeling the behaviors exhibited by some real-world concept or thing. These functions are often invoked by other objects, so they may also be thought of as implementing the modes of interaction between objects in a program. The requirements for these functions must be known before the class can be designed. There will often be a one-to-one correspondence between required behaviors and member functions to implement those behaviors. Maintaining this as an invariant makes it very easy to trace code back to requirements.

The set of application behaviors for a class should form a cohesive set of operations that are complete enough to accurately model the abstraction the class embodies and are only loosely coupled to any other code. This is where you most need to remember the three C's of good design: cohesion, completeness, and coupling.

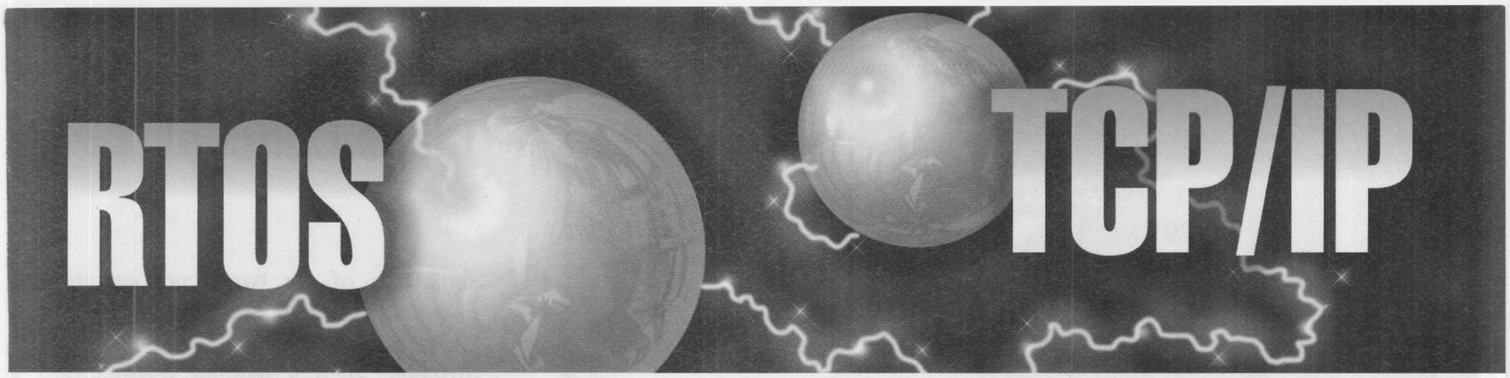
ACCESSOR FUNCTIONS

Public data members are almost never a good idea. In fact, public data members are almost always a *terrible* idea. In general, anything outside the class should have little reason to ever access internal class data. However, in those rare cases where such access is necessary, the appropriate way to provide it is through accessor functions that allow the class itself to fully control the access.

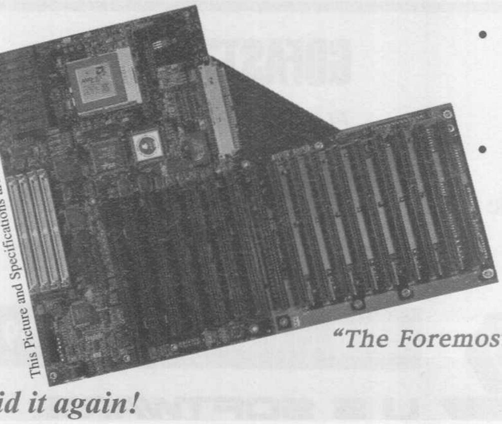
For example, let us assume that we have a class that has an attribute of type `Date`. We can provide the following accessor functions:

```
Date MyDate() const; // read my date
void MyDate(Date d); // write my date
```

Accessor functions are often called "getters" and "setters" and it isn't uncommon to see the above pair of accessor functions named **`getDate()`** and **`setDate()`**. However, because the


RTOS
TCP/IP

USNET



signatures of the two functions are obviously different, and the contexts in which they are used are also obviously different, such a naming scheme is completely redundant.

Note the `const` qualifier on the `read` function. This is always appropriate for functions which only read a value and do not modify the state of an object. The presence of the qualifier communicates both to a user of the class and to the compiler that this function can be invoked for either `const` or `non-const` class objects, and that it will not modify the object on which it is invoked. This not only helps the compiler to prevent programmer errors, it also helps the optimizer to know when it can be more aggressive.

Now, if instead of providing these accessor functions we just had a public data member, then we would have no control over who, how, or when our date was modified. And if we ever wanted to change the name of that data member (say when adding another `Date` member and wanting two different names), or if we decided that the `Date` could be calculated instead of stored, then every piece of code that accessed that data member would have to be modified. These are all very bad things.

By using accessor functions instead, we have provided ourselves a great deal more control and flexibility, both of which can be exercised without affecting other code. For instance, if both of these accessor functions are provided and we decide that the date should be calculated rather than stored, we can implement this change without affecting any other code. We could also choose from the beginning to provide only the read accessor function and make our date read-only. Inside the class, we could still set the date as appropriate, but no other code would have write access. Further, we could choose to provide only the write accessor function and have a write-only date that no other code could read back.

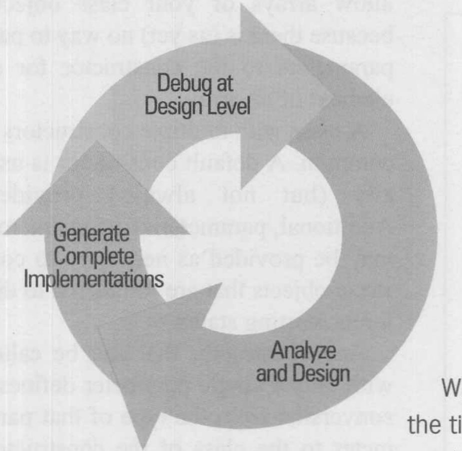
The point is, by decoupling the internal representation from the inter-

BE WRONG MORE OFTEN

And then, be right when it counts.

The path to design perfection requires rigorous exploration. And knowing early on what won't work is a sure way to find out what will.

But there's the age-old conflict of embedded systems software design. You have a deadline. Do you take the time to plan, analyze and explore, or succumb to the rush to code?



Spend more time designing with Rhapsody.

Rhapsody™

What if there were a way to "freeze the clock" and take all the time you need to plan and validate your design–actually *see* how it behaves before it's completed? What if you could test every concept? Explore new ideas?

What if there were a way to make a change to your design and *instantly see* it reflected in your implementation? Or make a change to your implementation and *instantly see* it reflected in your design? What if there were a way to do all of this, and get full, optimized and targeted production-quality C++ *automatically?*

The software Holy Grail? Yes. **Introducing Rhapsody, the first UML-based OO software design and implementation tool.** Rhapsody will remarkably change the software development paradigm from time spent coding to time spent designing.

Save time and money. Inquire about Rhapsody now. For a special, limited time offer and white paper, *"The Next Generation: Modeling with UML,"* contact us toll free at 1-888-8 ILOGIX (1-888-845-6449) ext. 100, or email info@ilogix.com and get the score on Rhapsody.

REAL-TIME OBJECT-ORIENTED TRAINING SERIES

The only series of its kind:
OO Analysis & Design, and Software Scheduling

i-Logix

Three Riverside Drive • Andover, MA 01810 • 508-682-2100 • info@ilogix.com • www.ilogix.com

CIRCLE # 36 ON READER SERVICE CARD

face to that representation, a great deal is gained. Usually most simple accessor functions can be made inline, so that this control and flexibility can be gained without even paying the price of a function call. The moral of the story is never to allow direct access to class data members, and that accessor functions are good and direct access is bad.

CONSTRUCTORS

A class constructor is called to initialize a newly created object to a known good state. Constructors are never invoked directly. Instead, the compiler ensures that the appropriate constructor is called whenever an instance of a class is defined, or when an instance of a class is allocated via the new operator. In either case, the compiler ensures that storage for the object is already allocated prior to calling the constructor, and that the `this` pointer is already set to point to that storage.

A constructor that takes no parameters, or one that can be called with no parameters because it has default values for all of its parameters, is called a *default constructor*. A default constructor is necessary if you wish to allow arrays of your class objects, because there is (as yet) no way to pass parameters to the constructor for an element of an array.

A class with multiple constructors is common. A default constructor is usually (but not always) provided. Additional, parameterized constructors may be provided as necessary to construct objects that are initialized to different starting states.

Any constructor that can be called with only a single parameter defines a conversion from the type of that parameter to the class of the constructor. Note that this also includes constructors with more parameters that provide default values to allow calling with a single parameter. Such a conversion will be implicitly applied by the compiler when an instance of the parameter type is provided where an object of the class type is needed. This will occur without warning or complaint from the compiler. For example, given a class `C` with the following constructor defined:

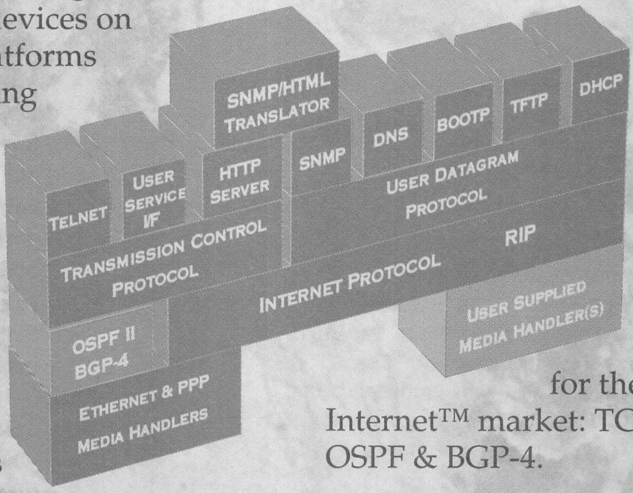
```
C::C(int val);
```

It becomes perfectly valid to pass an `int` to any function that expects an object of class `C`. The compiler will implicitly call the above constructor without warning or complaint, although this will clearly mask an error if the caller really meant to pass an object of class `C` and accidentally used the name of an integer variable instead. Indeed, this behavior is so dangerous that the ANSI C++ Committee has added a new keyword, `explicit`, that can be used to disallow these implicit constructor calls and force such con-

Q: What do pSOSystem™, Nucleus™, VxWorks™, Chorus™, & Your RTOS Have In Common?

A: Networking Solutions From Epilogue.

With over 3 million embedded solutions deployed, Epilogue's Envoy™ SNMP is the most-proliferated embedded agent available. Our architecture allows designers to deploy managed devices on CISC and RISC platforms from 8 to 64 bits using today's most-popular embedded operating systems, or your own kernel - all from one ANSI C source code set.



Now, Epilogue has extended the power of SNMP by providing a seamless interface for your customers to use any on WWW browser to configure and manage devices with Decorum™.

Decorum allows OEMs to build a customized GUI tailored to your device, while providing all of the power and flexibility of standard SNMP management - the best of both worlds. Complementing Decorum, Epilogue also offers a complete family of source code products for monitoring and connectivity solutions geared for the Embedded Internet™ market: TCP/IP, PPP, OSPF & BGP-4.

Find out what over 300 OEMs already know - Epilogue is your Source Solution for portable, compliant and supported network connectivity, monitoring and management.

EPILOGUE
Embedded Solutions
From Integrated Systems, Inc.

www.epilogue.com

CIRCLE # 38 ON READER SERVICE CARD

versions to be made explicitly. If your compiler supports this keyword, then use it liberally.

DESTRUCTOR

The class destructor (there can be only one per class) is called when an object is going out of existence. The destructor is called immediately prior to the deletion of the object's storage, so the `this` pointer is still valid in the destructor. The destructor should perform any cleanup actions that may be necessary. If the class is intended to be used as a base class for further derivation, its destructor must be declared virtual in order to ensure that the correct destructor is always called.

One of the most common uses of destructors is to free up resources that are acquired by the class constructor. For example, an object may require dynamically allocated memory which is provided by calling operator new in the constructor. The destructor in this case must make the corresponding call to delete that memory in order to prevent resource leaks.

OPERATOR FUNCTIONS

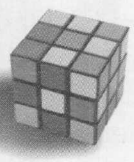
A C++ class may provide any or all of the following operator functions:

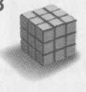
```
+, -, *, /, %, ^, &, |, ~, !, =, ==, !=, <, >, <=,
>=, +=, -=, *=, /=, %=, ^=, $=, |=, <<, >>, &&,
||, ++, --, ->*, ->, (), [], new, delete
```

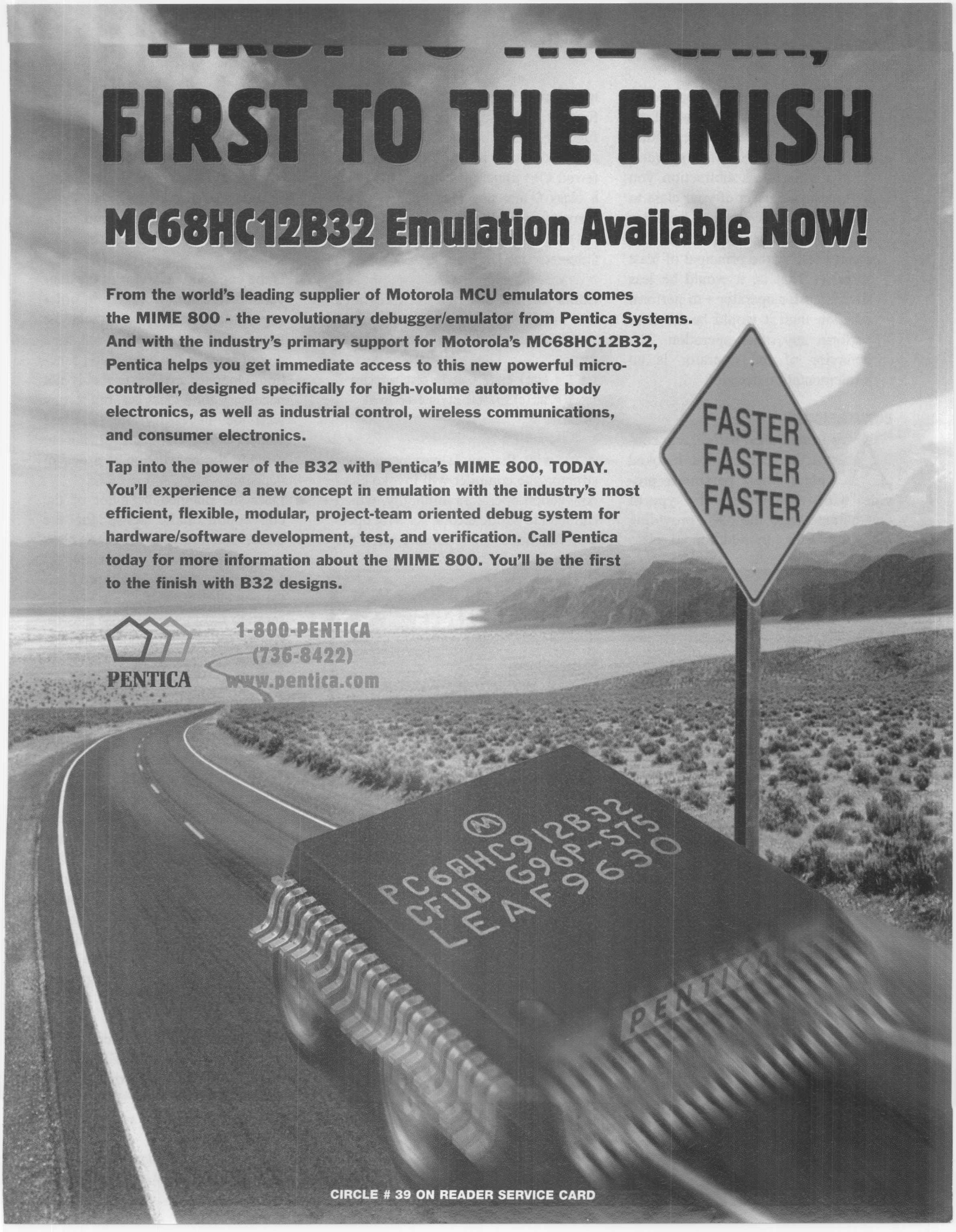
Obviously, not all of these operators make sense for all classes. How would you interpret operator `*=` for a class `Dog`? Also, there is nothing in the C++ programming language that forces a class designer to uphold the expected semantics of these operator functions. It would be perfectly legal in C++ to provide an operator `*=` for a class `Dog` and have that function reformat the computer's hard drive. It would also be perfectly legal in C++ to provide an operator `+` for an arithmetic type, such as a `ComplexNumber` class, and implement that function to perform subtraction instead.

It should be apparent from this discussion that the primary considerations when deciding about operator functions for a class are as follows:

- Is the expected behavior of the operator function necessary in order to provide users of the class with the functionality they need? If not, then do not provide the operator. Operator `*=` is probably not necessary for a class `Dog`, but it probably is necessary for a class `ComplexNumber`. Whether or not to provide the operator is a design decision
- Does the planned behavior of the operator conform to the user's most likely expectation of the operator's







FIRST TO THE CHIP,

FIRST TO THE FINISH

MC68HC12B32 Emulation Available NOW!

From the world's leading supplier of Motorola MCU emulators comes the MIME 800 - the revolutionary debugger/emulator from Pentica Systems. And with the industry's primary support for Motorola's MC68HC12B32, Pentica helps you get immediate access to this new powerful microcontroller, designed specifically for high-volume automotive body electronics, as well as industrial control, wireless communications, and consumer electronics.

Tap into the power of the B32 with Pentica's MIME 800, TODAY. You'll experience a new concept in emulation with the industry's most efficient, flexible, modular, project-team oriented debug system for hardware/software development, test, and verification. Call Pentica today for more information about the MIME 800. You'll be the first to the finish with B32 designs.

PENTICA

1-800-PENTICA
(736-8422)
www.pentica.com

CIRCLE # 39 ON READER SERVICE CARD

behavior? If not, then you can count on problems arising from use of the operator. No matter how carefully you document the fact that operator `+` actually performs subtraction, you will never get users of your class to expect this behavior. The behavior of any operator function should always follow the principal of least surprise. That is, it would be less surprising for operator `+` to perform addition than it would be for it to perform any other operation. The behavior of the operator is an implementation decision

CONVERSIONS

As I mentioned before, any constructor that can be invoked with a single parameter provides a conversion from the type of that parameter to the class providing the constructor. In addition, a class may provide any other conversion functions that are necessary. These functions may be used in either the standard C cast notation or in the preferred C++ functional notation. Given a class `C` that provides the following conversion function:

```
C::operator int();
```

Either of the following expressions will invoke the class-provided conversion operator on the object `c` of class `C`:

```
int i = (int) c; // C-style cast notation
int i = int(c); // C++-style functional
 notation
```

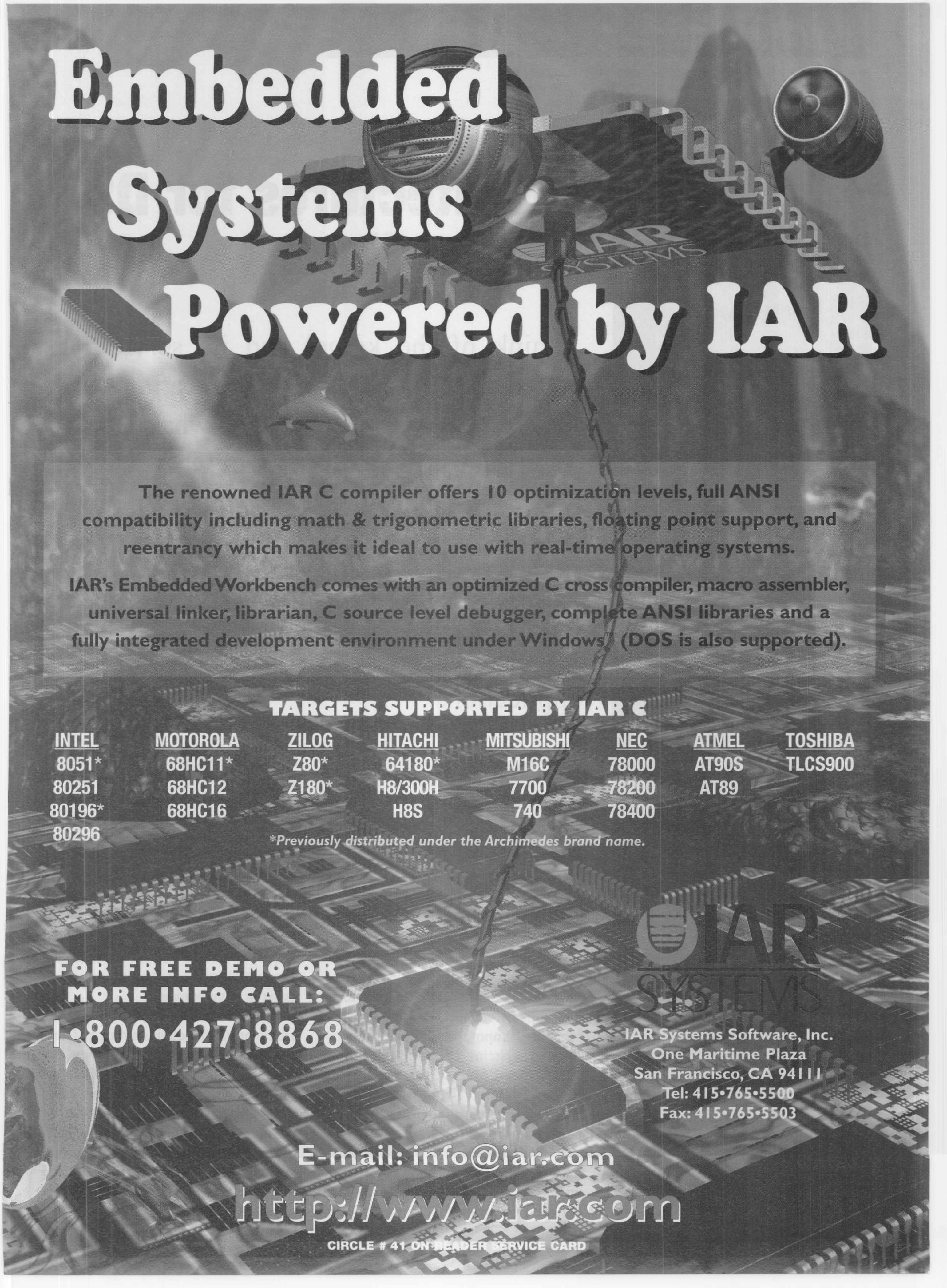
As with the single-parameter constructor, the compiler will invoke class conversion operators implicitly, which can lead to subtle errors. As with operator functions, C++ doesn't force the conversion operator to make sense or to be implemented in any reasonable manner. It would be perfectly legal in C++ to provide `Dog::operator int()`, but how much sense would this make? Thus, considerations similar to those for operator functions apply to decisions about conversion operators:

- Is the expected behavior of the conversion function necessary in order to provide users of the class with the functionality they need? If not, then do not provide the conversion function. Operator `int()` is probably not necessary for a class `Dog`, but it may well be necessary for a class `ComplexNumber`. Whether or not to provide the conversion is a design decision
- Does the planned behavior of the conversion make sense for the class? Converting any numeric class to an `int` is probably reasonable, but a conversion operator between class `Cat` and class `Dog` is unlikely to make sense to most users of those classes. Again, follow the principle of least surprise

POWERFUL AND FUN

There you have it. Classes in C++ are fairly straightforward and powerful extensions of structures in C. Writing classes in C++ involves writing a fair amount of code to assist the compiler in working with objects of your class, but careful attention to detail can have a big payoff in creating objects that are useful abstractions. The key is to focus on one aspect at a time, rather than confusing the issue by trying to think of everything at once. C++ was designed to help make programming more fun, so get out there and enjoy it. **ESP**

Randall A. Maddox is a consulting software engineer in the Washington, DC area with 14 years experience in C++, distributed systems, client-server applications, embedded systems, real-time control, security, and the like. Reach him at r.a.maddox@ieee.org.



The renowned IAR C compiler offers 10 optimization levels, full ANSI compatibility including math & trigonometric libraries, floating point support, and reentrancy which makes it ideal to use with real-time operating systems.

IAR's Embedded Workbench comes with an optimized C cross compiler, macro assembler, universal linker, librarian, C source level debugger, complete ANSI libraries and a fully integrated development environment under Windows® (DOS is also supported).

TARGETS SUPPORTED BY IAR C

INTEL	MOTOROLA	ZILOG	HITACHI	MITSUBISHI	NEC	ATMEL	TOSHIBA
8051*	68HC11*	Z80*	64180*	M16C	78000	AT90S	TLCS900
80251	68HC12	Z180*	H8/300H	7700	78200	AT89	
80196*	68HC16		H8S	740	78400		
80296							

**Previously distributed under the Archimedes brand name.*

FOR FREE DEMO OR MORE INFO CALL:

1•800•427•8868

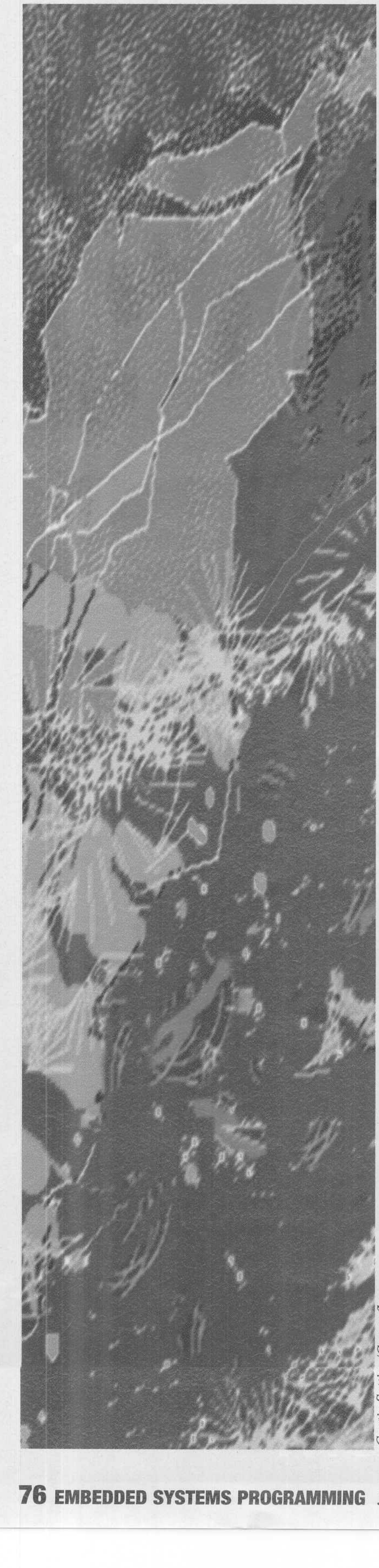
IAR SYSTEMS

IAR Systems Software, Inc.
One Maritime Plaza
San Francisco, CA 94111
Tel: 415•765•5500
Fax: 415•765•5503

E-mail: info@iar.com

http://www.iar.com

CIRCLE # 41 ON READER SERVICE CARD



Curtis Settino/Canoofle

Architectures for DSP Applications

Think of the kernel and its services as being distinct from the application. This context distinguishes between what does the processing on the system and what supports it, forces you to consider the appropriate architecture to support the application, and helps you decide how to make a custom kernel. This article presents several kernel architectures suited to different types of DSP applications.

DSP applications are generally different from most embedded applications, in which you can usually count on the services of a general multi-priority kernel. In the DSP world, though, *kernel* may be a foreign term, even though every application relies on some foundation to provide CPU resources, handle interrupts, and provide communication mechanisms. Full-featured kernels and operating systems aren't usually considered because of the overhead they impose on the tightly constrained systems that are typical in DSP. Instead, DSP software designers largely create their own supporting framework, albeit a reduced version of it, as a natural part of getting their system running within the product objectives and the limited CPU/memory resources available. Designers may not even be aware that in the process they have provided their own kernel services.

It is, however, useful to think of the kernel and its services as being distinct from the application and/or algorithm. This context draws a dividing line between what does the processing on the system and what supports it, forces you to consider the appropriate architecture to support the application, and helps you decide how to make a custom kernel and seek alternative tools to assist in development.

With this distinction in mind, this article briefly discusses a range of kernel architectures and how they are suited to different types of DSP applications, starting from the most basic incarnations and building on these to illustrate more complex and generalized schemes. Whether you're writing your own kernel or getting outside assistance, the considerations in kernel selection and use are the same—effi-

In the DSP world, *kernel* may be a foreign term, even though every application relies on some foundation.

ciency, compactness, and simplicity vs. flexibility, expandability, and security.

From simple to relatively complex applications, the framework underneath may take one of these general forms:

- Single task with an interrupt service routine providing I/O
- Multiple tasks sequenced with co-operative yield
- Multiple execution threads time-sliced in round-robin fashion
- Combination: a number of co-operative task sequences time-sliced in round-robin fashion
- Preemptive multi-priority scheme

SINGLE TASK WITH ISR

Figure 1a depicts a relatively simple arrangement where a task processes a buffer of data that is collected from the outside world at a specific sample rate by an ISR (Interrupt Service Routine). The ISR is triggered by the arrival of samples from a peripheral such as an analog-to-digital converter that is connected to the DSP chip. Even this scheme requires a significant number of kernel services to be implemented before you can actually get the system up and running. We'll describe them in some detail, because together they cover the basics of kernel operation.

First, the system requires an initialization that enables the appropriate interrupt and sets-up the peripheral handling I/O so that it works at the right sampling rate and triggers an appropriate CPU interrupt. If the DSP chip doesn't support shadow registers (whereby CPU registers are pushed onto a special "shadow stack" before entering the ISR, and popped back off upon exit), the ISR framework will have to first save the values of any registers used within the routine so they can be restored at the end. That allows the interrupted task to continue unaffected upon return. These details are small but not insignificant; ISRs must be coded with precision because the smallest error can cause incorrect context management, resulting in a bug that can take days or weeks to find, due to the difficulty of reproducing it.

Perhaps more interesting is the communication between the ISR and the task. Typically, the ISR handles data one sample per interrupt, while the task may need a buffer of samples to process. In the case of an ISR collecting input samples for the task, its job is to add a sample to a buffer upon each interrupt, notify the task when it's filled the buffer, and get a new buffer, so that subsequent interrupts save samples to a different array. Meanwhile the task's processing loop would look something like this:

- Wait for a full buffer from the ISR
- Process the buffer
- Free up the buffer so the ISR can re-use it to collect samples, then return to the top of the loop to wait for more data

In order to maintain real-time data throughput, the system would have to allocate space for at least two buffers. Then while the task is processing one buffer, the ISR could continue to collect samples in another. An exchange mechanism may be handy as well, so the ISR could pass the task full buffers, and the task could send buffers it's finished with back to the ISR to be reused.

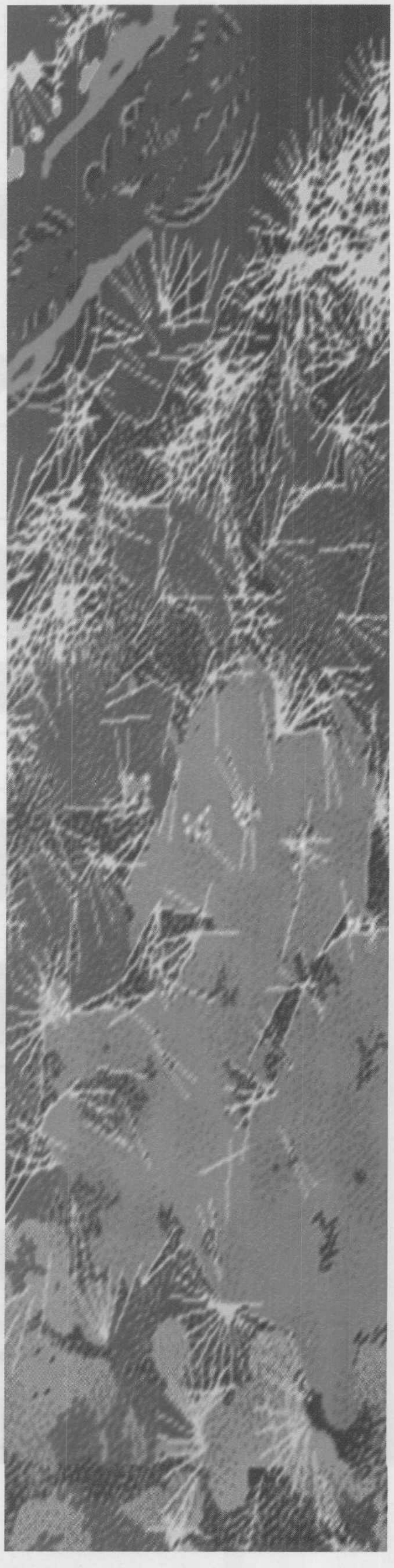
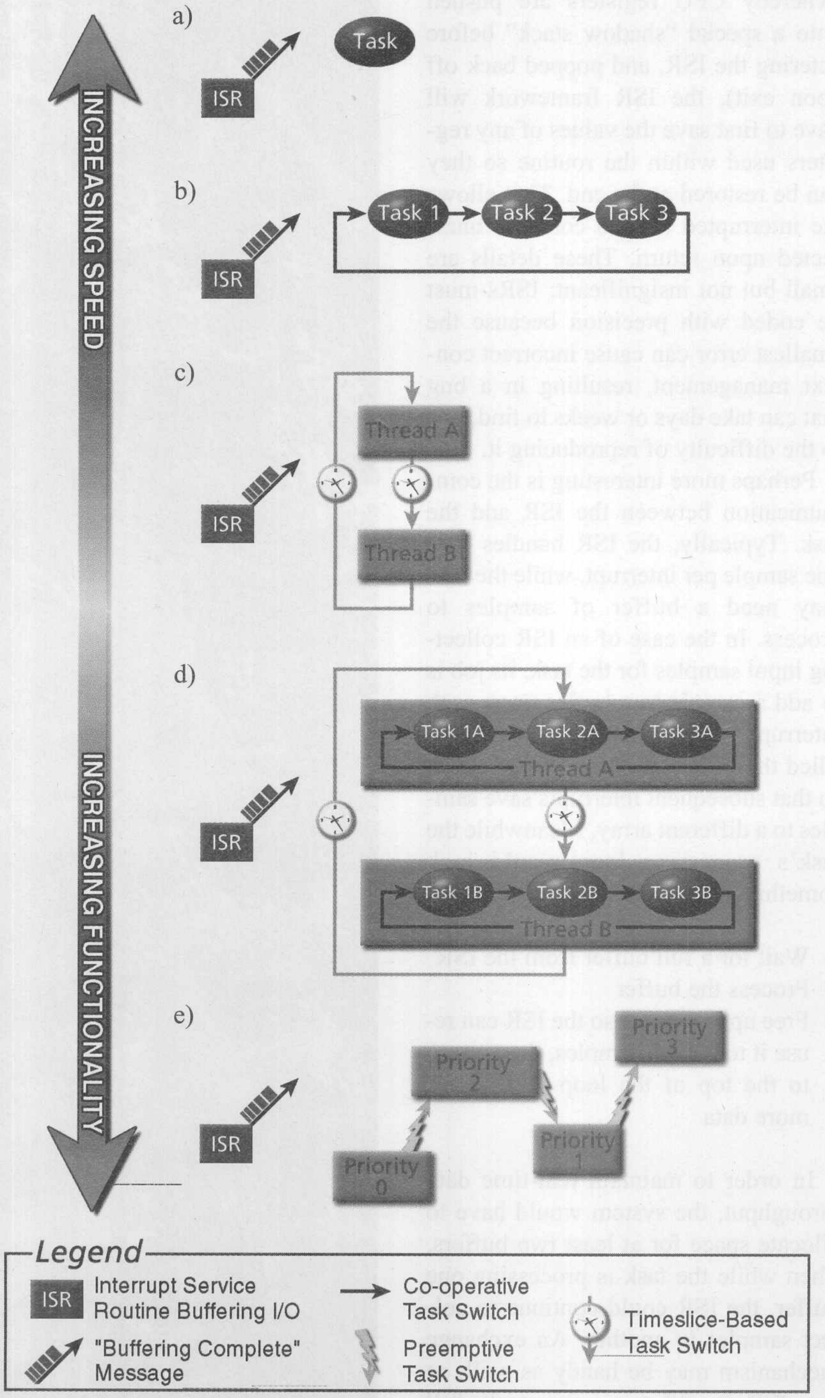


FIGURE 1
ISR examples.



Synchronization refers to how the task waits for a data buffer to be ready. The simplest way is for the task to poll a flag until it indicates that a buffer is ready; however, this implies that the system will never do anything more than run that one task, consuming all CPU (except during interrupt handling) whether it is actively processing or not. For an application that will only have one processing channel and stage, though, this scheme does the job very efficiently.

MULTIPLE TASKS

A more general approach involves using a synchronization mechanism instead. The task can then be suspended while it waits for a signal from the ISR. Until this signal arrives, the task would consume no CPU cycles, thus freeing processing power for other potential tasks or channels of processing. Using a conventional kernel object, the signaling could be achieved via a *semaphore*—a representation of a resource that can be owned by only one process at any given moment. While filling the buffer, the ISR would own the semaphore and release it when filled, thus allowing the processing task to take ownership of the semaphore.

This scheme implies that you have a kernel that can suspend a task seeking a used resource and can generally manage multiple tasks on a single CPU. The kernel allows you to expand the system just by adding tasks to the system list, but how this expansion is done depends on the application and in turn on the type of task-switching mechanism used by the kernel.

These issues are discussed in the remaining examples, but before moving on, let's remind ourselves that this simplest of systems has all the basic elements of a kernel: initialization sequence, interrupt handler for I/O, inter-task synchronization/communication, task manager to direct task execution, and memory management for creating the signal buffers.

COOPERATIVE SEQUENCING

Let's say we need to expand the single task with an ISR example to do more processing stages. The application might be a multi-featured digital answering machine with tone detection, speech recognition, and voice compression (recording) on the same input signal. As Figure 1b illustrates, each processing function can be performed by a separate task and each task can then be scheduled in a sequence. Once the first task has received a buffer from the ISR, it will run to the completion of its processing duties. Then it will explicitly give up the CPU to the next task, which in turn will process to completion and give the CPU control to the next task, and so on for all tasks in the system.

This scheme in which each task voluntarily yields the CPU to the next requires a kernel capable of *cooperative multitasking*, one which maintains a list of all tasks and the order of execution but doesn't do any task switching until explicitly called upon by a relinquishing task. The advantage of this set-up is that task switches can be optimized to take minimal overhead. Because you control when the task switches occur, you can determine which CPU registers must be saved before yielding and which ones don't matter. Minimizing this context switch time can be important in achieving good performance (reduced response time to interrupts which may be disabled during the switch, and fewer memory resources required). The chief disadvantage, as users of Windows 3.x are well aware, is that if one of the tasks runs out of control or gets stuck in a loop, it will take down the entire system with it—no other means are available for regaining control of the CPU.

Another application suited to the cooperative scheme (because a specific task order can be maintained) is one with multiple processing stages such that each successive stage depends on the previous one having been completed. A cellular phone with its various filtering and decoding stages is such an example.

Due to the cooperative scheme within each channel, one task going bad can affect the entire channel—hence the need for round-robin scheme.

In some ways, this arrangement is very similar to adding extra processing routines within the main task of the example in Figure 1a; however, it's convenient to package them as separate tasks so they can be deployed and developed as separate units and therefore dropped into different schemes and combinations.

ROUND-ROBIN SCHEME

A safer alternative to the example in Figure 1b is to have a different task scheduling scheme that makes each task's execution less dependent on the others. This alternative requires a time-based task switcher (see Figure 1c) which causes execution control to switch to the next task in the system list after the current one has run for a certain amount of time (the task's timeslice).

Round robin is the common description for this scheme, because each task gets an equal opportunity to run. Some tasks may still be more equal than others, however—while the timeslice is a system-wide parameter in some kernels, it could be designed so that each task has its own timeslice value. This would be a simple way of giving a specific proportion of the CPU to each task.

Round robin is safer than cooperative multitasking because the task switches don't rely on a voluntary yield of the CPU made by the relinquishing tasks; instead, they occur automatically upon the timeslice expiration. With the CPU divided into separate independent slices, the system can be said to have multiple, separate execution threads.

The cost of this security is that you don't know exactly when switches will occur, so the accompanying context switch requires saving the entire set of CPU registers used by the thread to guarantee preserving the environment of the current thread and restoring the entire CPU environment for the next one (saved when it was switched out). While a full context save/restore may be acceptable in some situations, switch times in the DSP world must usually be fast (sub-microsecond times are required to allow an effective response to an interrupt which may happen at a rate of tens or hundreds of KHz, and in general to save precious CPU cycles).

ROUND-ROBIN THREADS AND COOPERATIVE SEQUENCING

You can combine kernel schemes in a straightforward way. Suppose you want to expand Figure 1B's cooperative system to handle multiple channels because a replacement chip with more MIPS and memory capacity suddenly became available. You could package each cooperative task sequence into an execution thread and replicate the thread for each additional channel you wanted. Each thread would be given its own CPU timeslice so that execution would proceed from one thread to the next in round-robin fashion. Because there's a cooperative scheme within each channel, one task going bad would affect the entire channel. However, with round-robin switching at the channel level, the bad channel wouldn't necessarily disrupt operation in other channels.

This scheme (see Figure 1d) is thus able to handle a relatively complex application, and it's easy to scale it up. Simply add another execution thread and give it a timeslice, as long as there are sufficient CPU cycles and memory space to schedule the thread and still make real time. This approach is useful for any application in which multiple channels are doing similar things, such as a multi-line voice mail system, a wireless base station, a PBX, a polyphonic music synthesizer, and so forth.

To support this architecture, the kernel must be able to maintain a list of threads that are switched in round-robin fashion, and simultaneously maintain a cooperative task sequence within each thread. While it's the fastest and most flexible scheme yet, it lacks the complete generality of a full-featured multi-priority kernel, which may be needed in certain applications.

PREEMPTIVE MULTI-PRIORITY SCHEME

For those applications that need the most flexibility, taking the final step in kernel or operating system implementations may be required—using a preemptive multi-priority kernel. However, this scheme is more popular in large floating-point DSP systems and very rare in integer DSP systems.

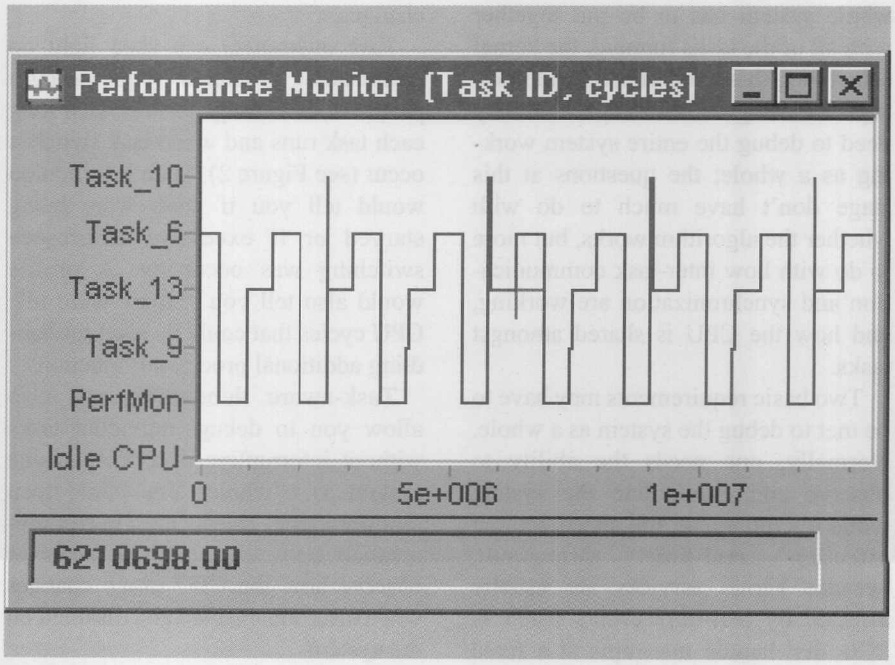
In this case, scheduling is done according to priorities: the "ready" task with the highest priority assignment is always executing. A task becomes ready when any resources (messages or synchronization signals) that it may have been waiting for are now available to access. If a lower-priority task happens to be executing when this occurs, it will be preempted in favor of the higher one, regaining execution time only if all higher-priority tasks are blocked.

If multiple tasks have the same priority level, then as long as execution is maintained at that priority, it will occur in round-robin fashion. Thus the multi-priority architecture can be viewed as an extension of the schemes depicted in Figures 3 and 4, through the replication of the round-robin sequence at each priority level on the system (see Figure 1e).

Multi-priority kernels, while flexible, require more care to use properly because the run-time execution sequence can be difficult to determine, especially if you allow tasks or threads to change priority dynamically. An excellent discussion of programming pitfalls from mismanaging priorities is presented by Bill Lamie ("Multitasking Mysteries Revealed," *ESP*, February 1997, p. 38), in whose article various ways of starving tasks and causing excessive context switching to occur are among the topics that are covered.

In addition to the programming complexities, a major concern for real-time DSP applications is the overhead—in terms of CPU execution time, code size and data space used—which can be dramatically greater than the other schemes. Managing multiple priorities requires a task scheduler that maintains multiple tasks/thread lists at each priority level and scans for the highest priority task to run whenever a synchronization event occurs. On small fixed-point systems that use every available CPU cycle, this scheduler can take a significant portion of the available resources. Thus its current unpopularity on integer DSP systems, most of which are highly cost sensitive (consumer items).

FIGURE 2
CPU usage profile.



Despite these drawbacks, a multi-priority scheme is ideal for certain situations. For example, a mixture of time-critical (high priority) and background (lower priority) tasks on the system lends itself to this scheme. Some integrated cellular phone applications are like this, where the DSP is used to process both time-critical baseband signal data and respond to user keypad input which can tolerate a relatively slower response time. As DSP and microcontroller functionality continue to merge, more and more applications fit this arrangement.

Some speech recognition algorithms consist of a time-critical speech analysis portion and a background pattern matching process. Also, in the case where some tasks process smaller signal buffers than others, giving a higher priority to the smaller block processing can ensure that it gets its job done on time. In today's complex systems that can combine algorithms from different sources, seeing a mixture of signal frame sizes is common. For example, a tone detection algorithm that processes 20ms of a signal at a time may work beside a speech detection that requires 8ms buffers.

We mentioned context save size during task switching as a factor affecting performance. Scheduler-induced interrupt latency is another. Task switching adds to the time required to respond to an interrupt, because interrupts are usually disabled when task lists are accessed and manipulated. Therefore, the more complicated the scheduling scheme, the worse latencies usually are. In the DSP world, with real-time interrupts occurring at tens of KHz and up for voice, interrupt latency is a critical factor.

DEBUGGING KERNEL-BASED SYSTEMS

Using a kernel presents some special debugging challenges. You can step in non-real time to debug and characterize components individually, but at some point the whole system has to be put together with all of the tasks running, the kernel scheduling them underneath, and interrupts handling real-time I/O. You may need to debug the entire system working as a whole; the questions at this stage don't have much to do with whether the algorithm works, but more to do with how inter-task communication and synchronization are working, and how the CPU is shared amongst tasks.

Two basic requirements may have to be met to debug the system as a whole. Normally, one needs the ability to observe and manipulate the system while it's running at full speed without affecting real-time throughput, because kernel services are usually affected by real-time events (such as ISRs that handle interrupts at a fixed real-time rate and task-switcher that triggers on timeslices). This is essentially true *real-time debugging*.

The designer must now be a low-level optimizer for in-house code as well as a high-level systems integrator.

In a multitasking system, you need to know the context in which observations are made, or possess an awareness of the task that is running at the time.

The benefits of real-time debugging are evident in the ability to view the actual signals input to and processed by the system as at full speed as they occur. One can also make changes to the system while it's running and then observe the immediate effect of these changes.

Task awareness can shed light on multitasking problems by providing a profile of CPU usage to show how long each task runs and when task switches occur (see Figure 2). Such information would tell you if tasks were being starved or if excessive unforeseen switching was occurring. A profile would also tell you if there were idle CPU cycles that could be used for handling additional processing functions.

Task-aware debugging can also allow you to debug individual tasks without interrupting execution on the system as a whole. You could then monitor kernel objects used in communication between individual tasks or observe how the CPU usage changes when tasks are enabled and disabled on the system.

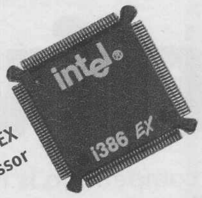
Achieving real-time task-aware debugging does add overhead to the system, and it also adds possibly greater effort in code development. In most cases, you'll steal CPU cycles by using a debug task to make observations. Sending the information to a host control/visualization environment also requires a link between the DSP and host that doesn't halt the DSP itself. Finally, task awareness and control requires some communication between the debugging system and the kernel. In the DSP world, as is the case with embedded toolsets, this communication can be achieved by having the kernel provide an interface to support basic services (memory read/write and task-level control) to a debugging environment. Again, as with kernels, the user can develop custom debugging utilities or choose to integrate with an available environment.

FITTING THE RIGHT KERNEL IN

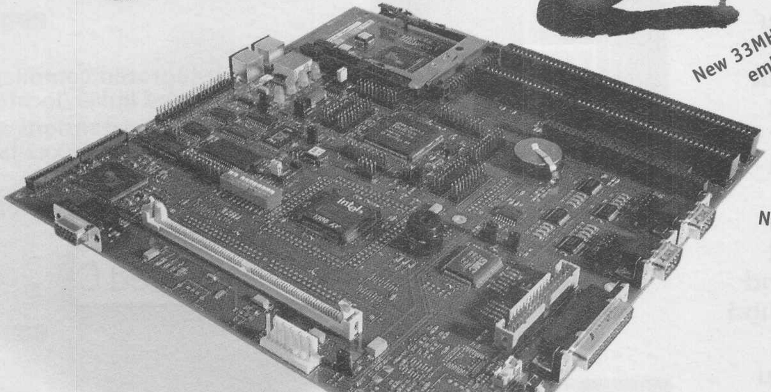
DSP software applications are getting more complex as they perform increasingly more functions that draw from outside components. Thus, the designer now has to be a low-level optimizer for in-house code as well as a high-level systems integrator. It is therefore important to consider not only functionality, but also the architecture most suited to support the various functions and algorithms that may eventually be running on the system. There are many ways to incorporate kernels into your system on your own or with outside help, each uniquely affecting the development and debugging process. The right choice will balance the need for services with the need to work within limited resources. ESP

Edmund Sim holds a Master's degree in Engineering in DSP from the University of Toronto. Prior to joining GO DSP as a Senior Software Engineer in the Development Tools Department, he was a lead DSP development engineer at Northern Telecom. He may be contacted at www.go-dsp.com.

radical technology embedded



New 33MHz Intel386™ EX embedded processor



QNX®

The Leading Realtime OS for PCs

QNX RTOS with Photon microGUI®

New RadiSys R380EX system controller

The New EXPLR2:

All the hottest embedded technologies on one handy eval board!

Combine the latest products from the leaders in embedded PC technology and what do you get? Faster time-to-market. More functionality with less hardware. A longer production life for your product. And a lot less work.

Intel's EXPLR2 makes your design work easy. All the hardware you need is fully integrated, so you can start prototyping right away. Better yet, the design-capture files come included. Just cut and paste to create your custom system.

The EXPLR2 also comes with QNX, the realtime OS that delivers high-end performance, even on low-cost hardware. Exceptionally small and fully scalable, QNX gives you a POSIX-certified environment *and* a full-featured GUI in under 1MB of memory!

The EXPLR2 comes with something else. Commitment. With their long-term commitment to embedded applications, Intel, RadiSys, and QNX Software Systems are the right companies to give your designs a real future. **Radical!**

For your free reference design kit, surf: www.explr2.com or call: 800 862-1883

(to order a board, contact your Intel distributor)

New

One-Stop Internet Toolkit!

Tools, demo apps, source code, hardware... ***everything*** you need to start **today!**

Intel product code: EX386INTRNETUS

CIRCLE # 43 ON READER SERVICE CARD

Intel and Intel386 are registered trademarks of Intel Corporation.
QNX and Photon microGUI are registered trademarks of QNX Software Systems Ltd.
RadiSys is a registered trademark of RadiSys Corporation.

Point, Click, Deliver

The competition is fierce. You are under pressure to get your product out fast and at the lowest cost. What you need is the INTERTOOLS KERNEL SOLUTION. It's the **only fully integrated 68xxx toolset with a Precise/MQX® kernel** and a host of easy to use features found nowhere else.

The Best 68xxx Kernel Solution

When it comes to helping you get to market faster – with a better built, better designed product – nobody beats TASKING.

No wonder TASKING is the world's leading choice for software development tools across industry standard computing platforms; Windows® 95, Windows® NT, UNIX. Throughout the world, more embedded applications have been developed using TASKING tools than anyone else's.

To find out how to deliver your product to market faster call us at **1-800-458-8276**, or visit us at **http: www.tasking.com**

TASKING

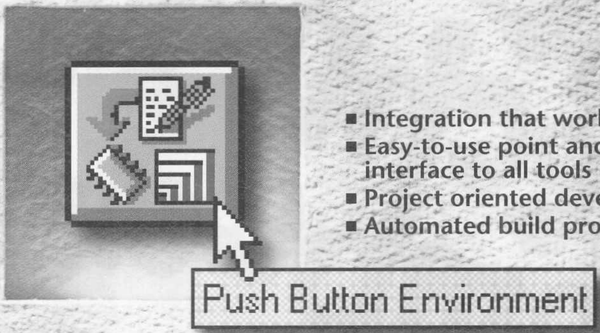
Quality Development Tools Worldwide

BSO/TASKING and Intermetrics Microsystems have joined to become TASKING

- Delivers significant cost savings
- Single and distributed processor operation
- Scaleable microkernel architecture
- Extensive embedded I/O components such as TCP/IP, PPP, LAPB, HDLC, CAN...

68302 Kernel

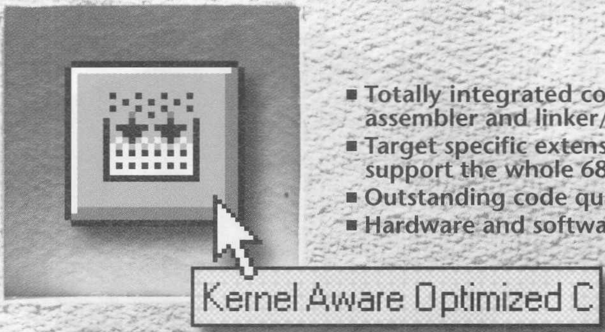
68040 Kernel



- Integration that works
- Easy-to-use point and click interface to all tools
- Project oriented development
- Automated build process

68EC000 Kernel

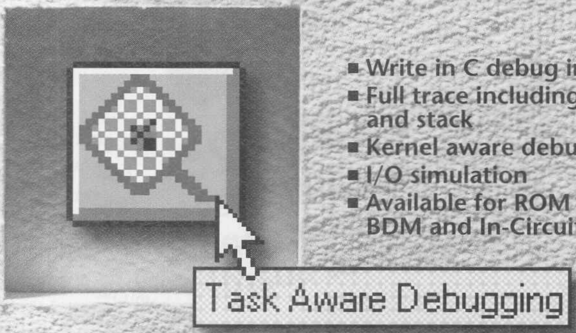
CPU32 Kernel



- Totally integrated compiler, assembler and linker/locator
- Target specific extensions and libraries support the whole 68xxx family
- Outstanding code quality
- Hardware and software floating point

68340 Kernel

68360 Kernel

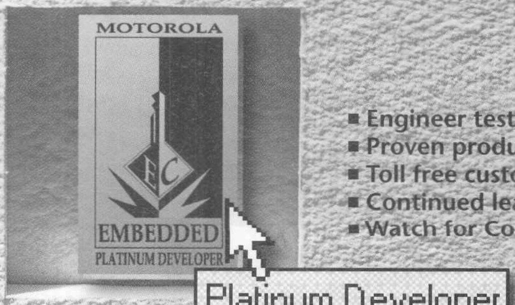


- Write in C debug in C
- Full trace including C, assembly and stack
- Kernel aware debugging
- I/O simulation
- Available for ROM monitor, simulator, BDM and In-Circuit Emulator (ICE)

68332 Kernel

68010 Kernel

68060 Kernel



- Engineer tested Motorola approved
- Proven product line since 1980
- Toll free customer support
- Continued leading-edge development
- Watch for ColdFire® and Power PC®

CIRCLE # 44 ON READER SERVICE CARD

Have 'em Your Way: 8- and 16-Bit MCUs/MPUs

by NICHOLAS CRAVOTTA

Even though the terms "8-bit" and "16-bit" are not the most explicitly descriptive, considering the number of different ways to define them, you may still find yourself asking the question, "How big a processor do I need?" The question is not whether your design requires a 16- or 32-bit processor per se, but rather that your design requires a certain level of performance. You don't want to pay for a 16-bit processor when an 8-bit will work just as well. However, today's processors are much more than just ALUs and data buses.

INTEGRATION

The true differentiator between processors is the list of extras — in other words, the on-board memory and other peripherals. Some families have over 100 members, each with a different amount of on-chip memory and configuration of peripherals. What's on the chip is important in that it defines the price you'll pay, the additional memory you'll need to place, and the amount of power you'll have to supply. Finding the right application-specific processor with the best price/performance ratio for your design means understanding what kinds of performance your design requires and what options are available for fulfilling those requirements. In other words, your application drives how your processor should be integrated with extras.

What does it mean for a chip to be application-specific? Controllers are shaped around the priorities typical for a particular type of application. For example, controllers for communication applications need good connectivity; that is, they possess communication interfaces and serial ports. These devices are characterized by low power consumption and voltage. If you're working on an industrial application, you probably don't have the high volumes necessary to justify spinning your own chip, so you have to look for a good match from among what's already out there. You'll need a robust part, one that can handle the harsh environments to which your application will find itself subjected. If your application is slated for the consumer market, then cost is probably your driving need. A 3-cent savings per processor could make the difference between profit and loss.

Automotive applications are high-volume, cost-sensitive applications. Because of these high volumes, auto manufacturers can drive spins of chips, effectively "suggesting" to chip manufacturers that a particular configuration would make a good standard part. Chip manufacturers can only spin off a new variant of a processor if there is enough overall volume to support it. Thus, standard processors often possess peripherals used by a majority of applications to increase the number of designs that can use them, thereby driving up the overall volume and dropping the price. If you can find a chip where you can use 80% or more of the chip's functionality, you've got a good match. High-volume customers, of course, can aim for near 100%.

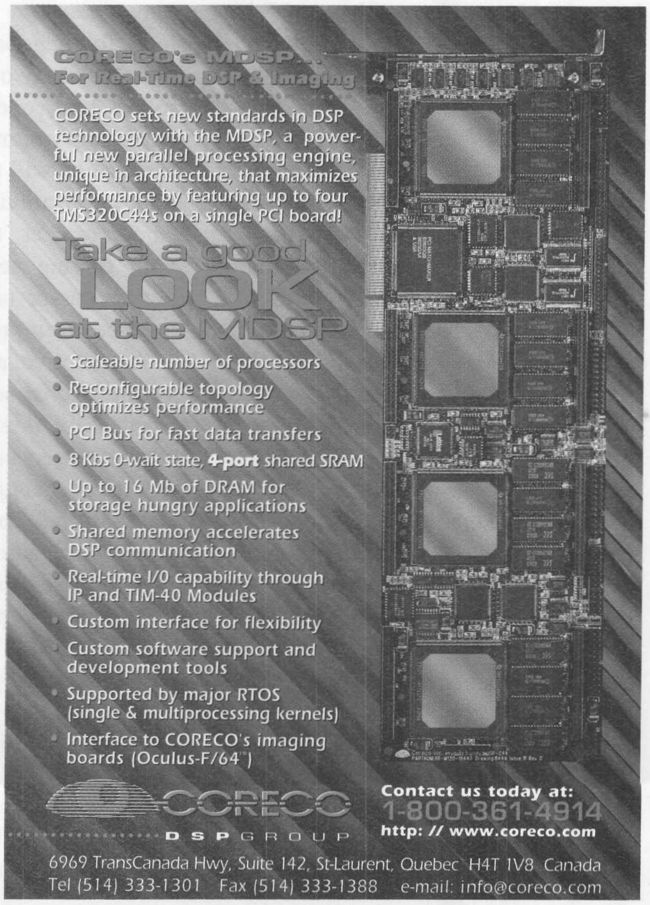
As more demanding applications arise, processors need to possess higher functionality and more memory. Even if your program isn't going to change, you'll still need more memory to control additional peripherals. As everything begins to communicate with everything else, serial and I/O ports become more important. Increasing use of signals, such as those from sensors, translates into increased on-chip analog-to-digital support. And as more data comes to a single point, more processing power is necessary to bear the load. You can implement your own peripherals in software, such as a serial port using I/O pins, but this uses memory, generates processing overhead, and can delay getting your product out the door. Paying for a peripheral will save you money in development costs.

Some peripherals define the application for which a processor can be used. For example, an on-board LCD driver costs enough in terms of silicon real estate and pin designations that using the processor for anything but an LCD application makes no sense. Picking the right kind of memory is important too. On-board memory is valuable because it offers faster access than external, or off-chip, memory. However, on-board memory is more expensive than off-chip memory, so buying the right amount is important. Cache can speed up data processing, but there is a point at which more cache merely adds to your system cost and not to the performance. Flash and EEPROM are useful for applications where code may change in the field, such as communication applications employing evolving protocol standards. One-time-programmable (OTP) memory is less expensive than flash or EEPROM, but you still have the ability to individually program parts with the latest software. Such flexible programming allows you to keep a design as generic as possible up until the programming stage, where products can be customized with different software. The least expensive program memory is masked ROM, at least on chips manufactured in high volume. This method is the least flexible of all, locking you into software and allowing for no revisions without paying for a new mask. A single software bug can leave you with an inventory of useless parts.

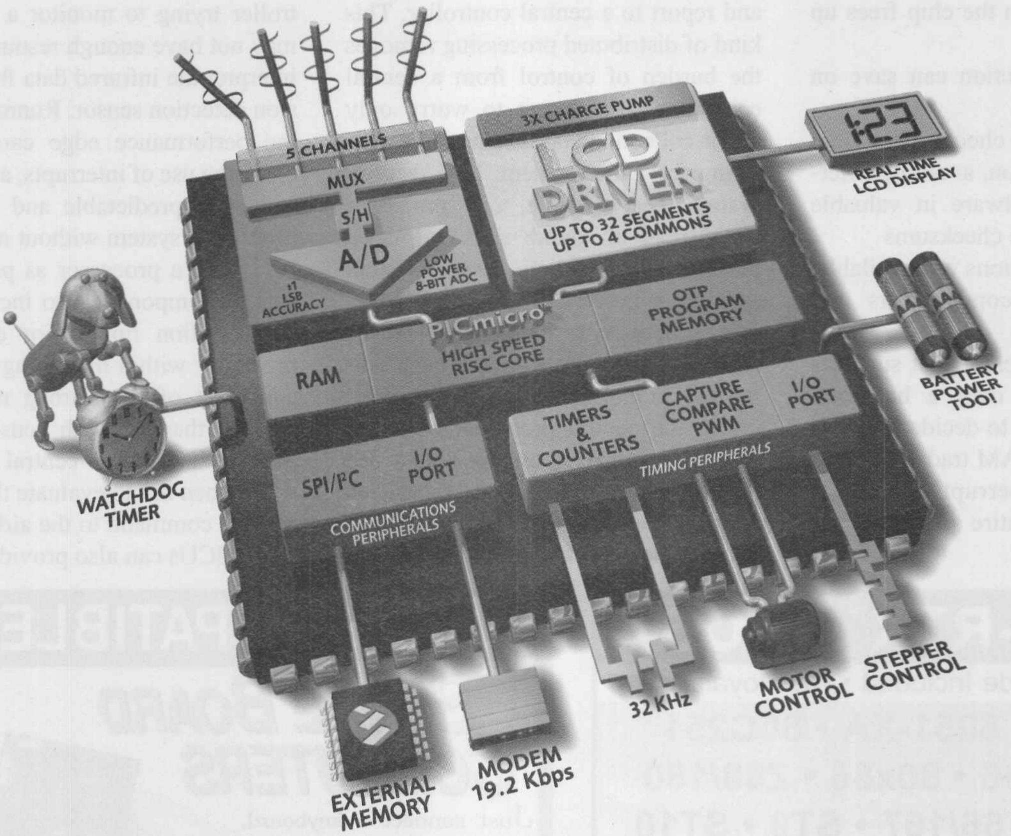
From where do all of these different varieties arise? The trend for chip manufacturers has been to step away from redesigning the entire chip each time they want to release a new variant. By using cores in conjunction with modular design architectures, components piece together, creating a whole new part with significantly less effort. The trick lies in selecting a configuration that will meet the needs of enough designs to generate volume sales and drop the price. Too many variants mean more engineering and support costs without necessarily higher overall volumes. Thus, buying a standard product with peripherals you don't use may cost less in the long run because of the high composite volume of sales. It's a better solution than if the chip manufacturers offered you the perfect part but at a lower volume, and thus higher, price. To reduce costs, a vendor may offer several variants that all come from the same silicon mask. For example, one variant may have an ADC and the other not. Both chips could be identical, but the ADC on the second chip remains untested and can thus sell for less.

An important consideration with peripherals is how they affect the pincount of the controller. Packaging can sometimes cost more than the silicon it houses, so make sure that the 20% of the peripherals you aren't going to use don't add unnecessary pins. You also need to consider how you will drive the peripherals, either by writing your own device drivers or using off-the-shelf drivers. Integrated peripherals don't necessarily supply the highest performance, focusing rather on function and general purpose rather than robustness. Here are other peripheral issues you should consider as well:

- ■ Interfacing to a peripheral should be possible without too much extra glue logic. For example, the LCD driver should contain a charge pump or you will have to supply the 7V that the LCD expects
- ■ Choosing a processor with an extra serial port opens the door to increasing your product's serviceability and providing diagnostic



Need an 8-bit OTP MCU that Drives an LCD?



Microchip makes it.

Introducing the PIC16C92X, the first cost-effective 8-bit OTP MCUs with an integrated LCD controller. How do you pack more punch into your design and get it to market before somebody else does? Turn to Microchip. We're taking one of our most feature-rich mid-range microcontrollers and adding an LCD interface for increased functionality. Think about it. When a single PICmicro™ MCU can do the work of two, you'll save space and money. What's more, the flexibility of an OTP MCU allows you to respond quickly to market demands and last minute design modifications. And with low-power features like the five-channel 8-bit ADC, the PIC16C92X MCUs aren't greedy with an LCD application's limited battery life. Whether you need to make your design better, smaller, faster, or simpler, Microchip makes it happen.

16 bytes LCD RAM • Up to 116 segments • Charge pump
64/68-pin packages or die • 500 ns instruction cycle • 4K x 14 EPROM

Die Available

MICROCHIP
The Embedded Control Solutions Company®
Microcontrollers • Non-Volatile Memories • ASSPs

Call us at 1-800-437-2767 or Visit us at www.microchip2.com/new/lcd

CIRCLE # 46 ON READER SERVICE CARD

capabilities through a dedicated port

- UARTs and PWMs have become fairly standard; they add to the pin count and may be unnecessary for simple I/O
- You can expect to see even more integrated interfaces, such as controller area network (CAN), in the coming year
- A reset circuit on the chip frees up one pin
- Brown-out protection can save on board logic
- On-chip parity checks for field memory corruption, avoiding external logic or software in valuable ROM to generate checksums
- Many timing options are available, such as timer coprocessors and watch dog timers
- Choosing a processor that supports a stack pointer over a hardware stack allows you to decide the optimal stack size/RAM tradeoff
- Finally, an interrupt controller changes your entire programming paradigm

CONNECTEDNESS

The availability of low-cost processors brings the dream of everything communicating with everything else closer to reality. For larger systems, this may include passing messages over the Internet. Within smaller systems, however, system components can control themselves and report to a central controller. This kind of distributed processing removes the burden of control from a central controller, allowing it to worry only about collecting status data, not monitoring the entire system. In a security system, for example, one processor could read all dumb sensors placed throughout a building. Such a system requires a powerful, and thus expensive, processor to handle the load of watching so many sensors. If each sensor had an inexpensive MCU monitoring it, the central processor could be much less powerful, only polling to see if an alarm condition was raised, as opposed to evaluating the sensor data to determine whether such a condition existed. Many controllers have integrated glue logic and analog-to-digital converters so they can connect to sensors directly. Additionally, placing the entire processing burden on a central controller can result in the controller running out of steam, thereby limiting the maximum size a system can grow before requiring an upgrade. A controller trying to monitor a car engine may not have enough resources to also interpret the infrared data from a collision detection sensor. Running close to the performance edge can also preclude the use of interrupts, as interrupts can be unpredictable and place high loads on a system without notice.

Having a processor as part of each system component also increases system reaction times. For example, a processor within the airbag subsystem could set off the airbag much more quickly than a dumb sensor sending information to the central controller, which then has to evaluate the data and sends a command to the airbag to activate. MCUs can also provide electrical

feedback to control systems and avoid some of the vulnerabilities faced by mechanical systems. An older garage door, for example, may check for too much force exerted by the motor to determine if it has squashed the cat. Controllers and sensors can use lights to check that the door is clear, monitor motor overload, and provide higher security. For some systems, statistical information can be valuable. A fire detector, for example, could keep an incident log for legal purposes, notify users when batteries are running low, perform self-tests, and provide internal redundancy.

The low cost of processors also promotes end users to adopt a "disposable technology" attitude. In many cases, manufacturing a new device is less expensive than repairing an old one, as is the case with alarm clocks and VCRs. While disposable technology raises environmental concerns, the fact that technology changes so quickly makes it more difficult for a competitor to steal intellectual property by examining the device. In the case of devices with security features, such as alarm systems or smart cards, thieves require more expensive tools to hack into the systems. In essence, reverse engineering a system costs too much and the technology will probably be stale by the time someone succeeds.

STATE OF THE MARKET

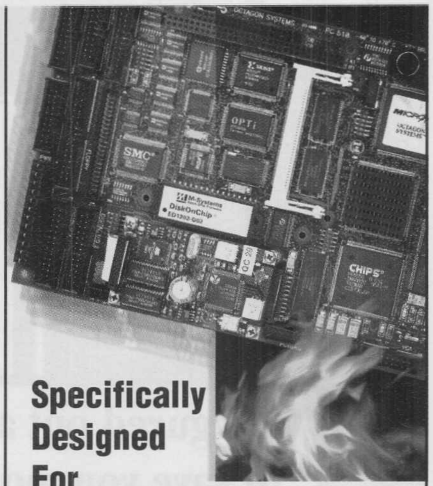
But isn't the 8-bit market dead? That's what we hear, at least, as the "world" migrates to 32 and 64 bits. Consider, however, that higher-end processors tend to cost a great deal more than a handful of lower-end processors. In 1996, 4-bit MCUs had an average sales price of $1.30, 8-bits were $2.99, and 16-bits and up, $6.99. According to Joyce Putscher at In-Stat, the 8-bit market is far from dead; it's growing and will continue to grow. Following only the dollar counts can be misleading, Putscher says, because although unit growth was good for these markets, a drop in the average sales price affected overall revenues.

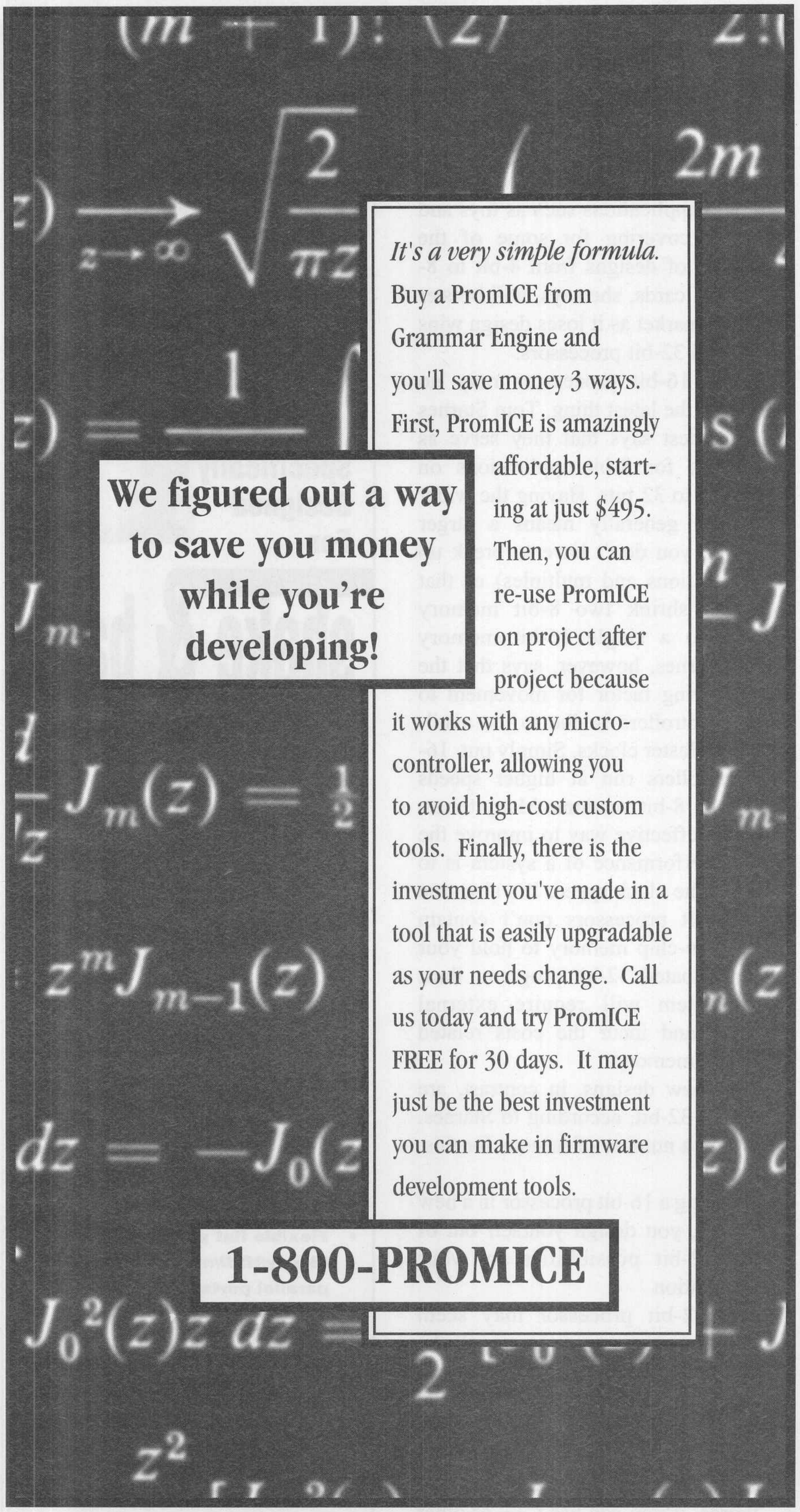
Putscher sees fairly flat growth in the 4-bit market, as MCUs find their way into applications such as toys and watches, covering for some of the migration of designs from 4-bit to 8-bit. Smart cards, she says, will bolster the 8-bit market as it loses design wins to 16- and 32-bit processors.

While 16-bit processors may no longer be the latest thing, Tom Starnes of Dataquest says that they serve as transitions for 8-bit applications on their way to 32 bits. Having the wider data path generally means a larger ALU (so you don't have to break up your additions and multiples) or that you can shrink two 8-bit memory fetches to a single 16-bit memory fetch. Starnes, however, says that the main driving factor for movement to 16-bit controllers is due to the availability of faster clocks. Simply put, 16-bit controllers run at higher speeds than most 8-bit versions. Many times, the most effective way to improve the overall performance of a system is to increase the clock speed. It's also true that 32-bit processors don't contain enough on-chip memory to hold your typical bloated 32-bit program; thus, your system will require external memory and incur the costs related with said memory.

Most new designs, in contrast, are primarily 32-bit, according to Starnes. He offers a number of reasons for this:

- By using a 16-bit processor in a new design, you design yourself out of the 32-bit possibilities for your application
- The 32-bit processor may seem more expensive, in terms of the processor itself and the extra memory your 32-bit application is going to demand, but there is the added consideration of the uncounted costs of design resources. In other words, if you can use C++ or other higher-level languages instead of assembly, you can reduce your development cycle and make up for the additional processor costs by getting to market sooner





■ Doesn't everyone want the latest under the hood? (I worked on a project in which we moved to a 32-bit procesor—not because we needed the performance boost, but because customers were buying from a competitor from whom they could get a "32-bit" system.)

It's difficult to tell which way engineers are leaning when they migrate from 8-bit to 16- or 32-bit processors. Often it depends on the application. Of course, if you wait long enough, you can go straight to 32-bit—in much the same way Napoleon dealt with his mail. Legend has it that he waited two weeks before opening his mail, figuring that within those two weeks, minor problems would have fixed themselves and the real problems would still be awaiting his attention. Waiting has its advantages, because most problems are minor; but it pays to remember that Napoleon also lost the war.

GETTING A LEG UP ON DEVELOPMENT

Specifying a controller isn't just about silicon. Certainly, the silicon determines important characteristics of the controller, such as whether the instruction set offers commands suitable for your application. For example, if every calculation has to go through the accumulator, math-intensive applications will expand in size. For smaller chips going into cramped spaces, you need to consider how you'll get a probe in there to see what's going on.

Getting up to speed quickly in a new architecture can save time and money. Many chip manufacturers offer seminars to walk you through new chips, occasionally charging a nominal fee to keep the college students out. Often you can get a development board, and application notes can give you a head start on developing firmware.

Developing device drivers for peripherals can be expensive and time-consuming. A number of manufacturers have begun to offer configuration tools for generating device drivers. For

TRACE 32®

ONE SYSTEM FITS ALL!

MEET US IN THE INTERNET! http://www.lauterbach.com

TRACE32-DEBUGGER

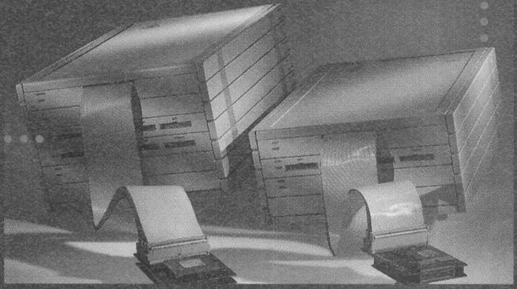
- Identical User Interface for all TRACE32 products
- Multitask-Debugger
- Directly loads more than 60 different file formats
- Full support for C++

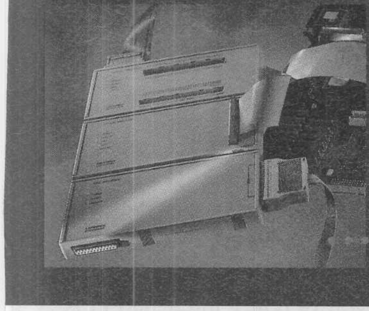
TRACE32-ICE

- Scalable
- Universal & Modular
- On-the-Fly Performance Analysis
- State and Timing Analyzer
- Up to 16Mbyte DUAL-PORT Memory
- Ethernet Option

TRACE32-ICECOMPACT

- Cost-effective solution for smaller projects



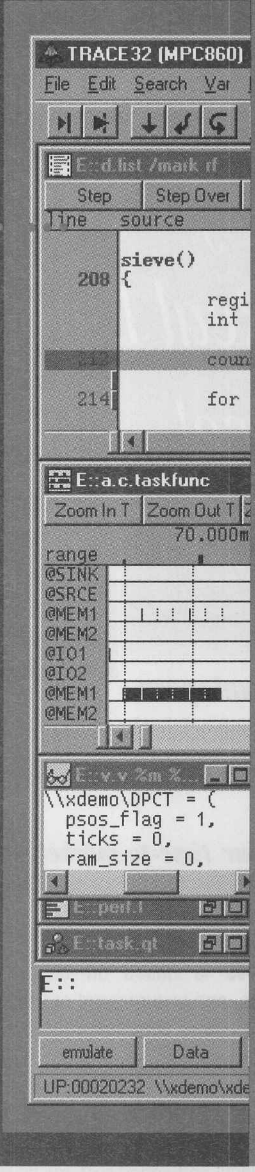


TRACE32-ICD

Our „Entry Level Solution"

- In-Circuit Debugger
- Early support of new chips
- Up to 1Mbyte Overlay RAM
- Ethernet Option

Easy to use with WINDOWS95



```
line   source
       sieve()
  208  {
               regi
               int
  212          coun
  214          for
```

```
        70.000m
range
@SINK
@SRCE
@MEM1
@MEM2
@IO1
@IO2
@MEM1
@MEM2
```

```
\\xdemo\DPCT = (
  psos_flag = 1,
  ticks = 0,
  ram_size = 0,
```

```
E::
```

```
UP:00020232 \\xdemo\xde
```

Hosts supported:

Windows 3.11, Solaris,
Windows-95, HP-UX,
Windows-NT, SCO-ODT,
VMS, DEC-Unix...

Open System:

RTOS aware:
HIOS, OS/9, OSE Classic, pSOS+, PXROS, RTC, RTX51/166, RTXC, VRTX,VxWorks...

Environments:
TORNADO, EasyCODE, Codewright...

New Chips:

AMD:	186EM, ES
ARM:	ARM7
Intel:	80C196 EA, KR, NT...
IBM:	Power PC PPC403
Motorola:	68HC05/08, 68HC12, 68EN302, 68308, 68328, MCF5102, Coldfire, MPC505, MPC821, MPC860
Philips:	51XA
Siemens:	C163 3.3&5Volt

...and more:

- 68000/1/8/10
- 68302/PM302/LC302
- 68306/307
- 68356
- 68330...336
- 68360, 68349
- 68020/030/040/060
- 68HC16
- 68HC11 A...-P
- 80186-EA...-EC, XL
- 80386SX/DX/EX/CX
- 80486SX/DX
- 80196...
- 8051...
- H8/500/300/300H
- 90C100...CL301
- SAB165/167, 80C166/GOLD
- Z80, Z180...
- NS32000
- MELPS 7700

CE

For more information please contact:

Lauterbach, Inc.
5, Mount Royal Ave
Marlborough, MA 01752
Fax: 508-303-6813
Phone: 508-303-6812

Lauterbach Datentechnik GmbH
Fichtenstrasse 27
D - 85649 Hofolding, Germany
Fax: ++ 49-81 04-89 43-49
Phone: ++ 49-81 04-89 43-0

e-mail info@lauterbach.com
http://www.lauterbach.com

LAUTERBACH

CIRCLE # 51 ON READER SERVICE CARD

Or call your sales office:

USA *508-303-6812, 617-320-94 00* ***Australia*** *++61-2-65 41-873* ***China*** *++86-10-2383376* ***Egypt*** *++202-349-59 31* ***India*** *++91-803-323029*
Israel *++972-3-6491202* ***Japan*** *++3-3405-0511* ***South Africa*** *++27-21-23-49 43* ***South Korea*** *++2-511-9846* ***Taiwan*** *++886-2-7057090*
Europe: Austria *++43-1-86 58 54 30* ***Denmark*** *++45-4344-6010* ***France*** *++33-1-39 89 96 22* ***Ireland*** *++353-1254-682092* ***Italy*** *++39-2-90 72 24 41*
Netherlands *++31-78-6816133* ***Spain*** *++34-3-291-7633* ***Sweden*** *++ 46-40-922425* ***Switzerland*** *++41-62 7710886* ***United Kingdom*** *++44-1254-68 20 92*

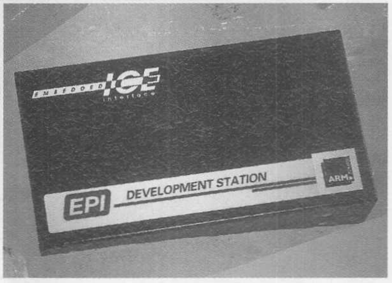
example, with such a tool you could configure a timer peripheral for four PWMs, select a frequency for each, and the configuration tool will kick out C-source device drivers in significantly less time than it would take for you to develop your own. This kind of tool is critical for making higher performance processors with multiple peripherals less complicated. Your application also becomes more portable, as you can port a device driver to another chip by having the configuration tool reconfigure your specifications for the other chip.

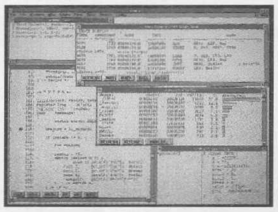
Throwing a little money at performance problems by choosing a slightly more powerful processor might save you time during development, instead of trying to squeeze a program into 2K or getting 2% more speed out of the chip. Picking a controller family with room for growth also helps, so you can further postpone the move to the next higher-bit architecture.

One argument states that x86 chips make excellent embedded processors. With the PC market driving volumes, development tools are inexpensive in comparison to traditional embedded tools. The big players invest heavily in their tools, so x86 tools are always state-of-the-art. Additionally, students self-taught on $100 tools are employable without much training. While there is some truth to this argument, it doesn't apply to much of the embedded market. C and C++ compilers optimized for RAM-bloated PCs don't fare well in the constrained environs typical of embedded systems. There is also the issue of how long 8- and 16-bit processors will be able to ride this wave, as the debate about whether to drop backwards compatibility continues. Finally, students writing code on PCs rarely ever see the x86 layer and will find the embedded world an entirely different and unforgiving beast.

In the last special report on processors ("Taking Off the Gloves: 16- and 32-Bit Processors," *ESP*, November 1996, p. 103), we took on many of the taxonomic challenges of grouping processors. One common distinction is whether a processor is an MCU or MPU (I've used these terms interchangeably in this article). Another distinction is whether 16-bit refers to data path, ALU, or external bus. It's best to look at the traits that define a specific characteristic. For example, it doesn't matter what the "16-bit" refers to if what you're really interested in is the ALU size. Our table of products (at www.embedded.com/97/sr9707.htm) focuses on the characteristics, not the labels. You can decide for yourself what's important and what isn't. ESP

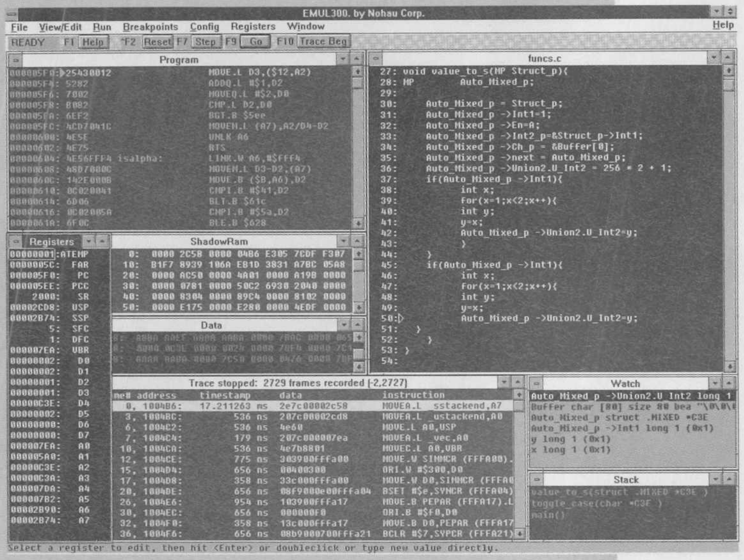
Nicholas Cravotta is the technical editor of Embedded Systems Programming. *He can be reached at ncravotta@mfi.com.*





nOHaU

In-Circuit Emulators & Background Debuggers



```
EMUL300. by Nohau Corp.
File  View/Edit  Run  Breakpoints  Config  Registers  Window                Help
READY   F1 Help   ^F2 Reset F7 Step  F9 Go   F10 Trace Beg

Program
000005F0:▷25430012                  MOVE.L D3,($12,A2)
000005F4: 5282                      ADDQ.L #$1,D2
000005F6: 7002                      MOVEQ.L #$2,D0
000005F8: B082                      CMP.L D2,D0
000005FA: 6EF2                      BGT.B $5ee
000005FC: 4CD7041C                  MOVEM.L (A7),A2/D4-D2
00000600: 4E5E                      UNLK A6
00000602: 4E75                      RTS
00000604: 4E56FFF4 isalpha:         LINK.W A6,#$FFF4
00000608: 48D7000C                  MOVEM.L D3-D2,(A7)
0000060C: 142E0008                  MOVE.B ($8,A6),D2
00000610: 0C020041                  CMPI.B #$41,D2
00000614: 6D06                      BLT.B $61c
00000616: 0C02005A                  CMPI.B #$5a,D2
0000061A: 6F0C                      BLE.B $628

funcs.c
27: void value_to_s(MP Struct_p){
28: MP          Auto_Mixed_p;
29:
30:     Auto_Mixed_p = Struct_p;
31:     Auto_Mixed_p ->Int1=1;
32:     Auto_Mixed_p ->En=A;
33:     Auto_Mixed_p ->Int2_p=&Struct_p->Int1;
34:     Auto_Mixed_p ->Ch_p = &Buffer[0];
35:     Auto_Mixed_p ->next = Auto_Mixed_p;
36:     Auto_Mixed_p ->Union2.U_Int2 = 256 * 2 + 1;
37:     if(Auto_Mixed_p ->Int1){
38:             int x;
39:             for(x=1;x<2;x++){
40:             int y;
41:             y=x;
42:             Auto_Mixed_p ->Union2.U_Int2=y;
43:             }
44:     }
45:     if(Auto_Mixed_p ->Int1){
46:             int x;
47:             for(x=1;x<2;x++){
48:             int y;
49:             y=x;
50:▷            Auto_Mixed_p ->Union2.U_Int2=y;
51:     }
52:     }
53: }
54:

Registers
00000001:ATEMP
0000005C:   FAR
000005F0:    PC
000005EE:   PCC
    2000:    SR
00002CD8:   USP
00002B74:   SSP
       5:   SFC
       1:   DFC
000007EA:   VBR
00000002:    D0
00000002:    D1
00000001:    D2
00000001:    D3
00000C3E:    D4
00000002:    D5
00000000:    D6
00000000:    D7
000007EA:    A0
000005A0:    A1
00000C3E:    A2
00000C3A:    A3
000007DA:    A4
000007B2:    A5
00002B90:    A6
00002B74:    A7

ShadowRam
  0:  0000 2C58 0000 04B6 E305 7CDF F307
 10:  B1F7 8939 106A EB1D 3831 A7BC 05A8
 20:  0000 AC50 0000 4A01 0000 A198 0000
 30:  0000 0781 0000 50C2 6930 2040 0000
 40:  0000 8304 0000 89C4 0000 8102 0000
 50:  0000 E175 0000 E280 0000 4EDF 0000

Data

Trace stopped:  2729 frames recorded [-2,2727]
me# address     timestamp     data                instruction
  0, 1004B6:  17.211263 ms  2e7c00002c58        MOVEA.L  _sstackend,A7
  3, 1004BC:        536 ns  207c00002cd8        MOVEA.L  _ustackend,A0
  6, 1004C2:        536 ns  4e60                MOVE.L A0,USP
  7, 1004C4:        179 ns  207c000007ea        MOVEA.L  _vec,A0
 10, 1004CA:        536 ns  4e7b8801            MOVEC.L A0,VBR
 12, 1004CE:        775 ns  303900FFFa00        MOVE.W SIMMCR (FFFA00).
 15, 1004D4:        656 ns  00400300            ORI.W #$300,D0
 17, 1004D8:        358 ns  33c000FFFa00        MOVE.W D0,SIMMCR (FFFA0
 20, 1004DE:        656 ns  08f9000e00FFFa04    BSET #$e,SYNCR (FFFA04)
 26, 1004E6:        954 ns  103900FFFa17        MOVE.B PEPAR (FFFA17).L
 30, 1004EC:        656 ns  000000F0            ORI.B #$F0,D0
 32, 1004F0:        358 ns  13c000FFFa17        MOVE.B D0,PEPAR (FFFA17
 36, 1004F6:        656 ns  08b9000700FFFa21    BCLR #$7,SYPCR (FFFA21)

Watch
Auto_Mixed_p ->Union2.U_Int2 long 1
Buffer char [80] size 80 bea "\0\0\
Auto_Mixed_p struct .MIXED *C3E
Auto_Mixed_p ->Int1 long 1 (0x1)
y long 1 (0x1)
x long 1 (0x1)

Stack
value_to_s(struct .MIXED *C3E )
toggle_case(char *C3E )
main()

Select a register to edit, then hit <Enter> or doubleclick or type new value directly.
```

68HC16 683xx

Real-Time Microprocessor Development Systems

—

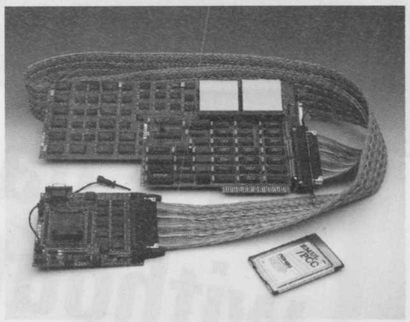
EMUL16/300™-PC

EMUL16/300-PC FEATURES:

- Supports both the 68HC16 and 683xx families.
- Hosted on PC's and Workstations.
- Modular hardware: low cost (BDM) to full feature.
- Real-time emulation at maximum chip speed.
- High-level C, C++ support for HIWARE, Intermetrics, Introl, Microtec, SDS, Sierra and more.
- In-line assembler and disassembler.
- Hardware and software breakpoints.
- Up to 4 Mbytes of emulation RAM.
- Trace board up to 512K deep, 104 bits wide, 40 bit timestamp.
- Triggering and filtering with full pipeline decoding.
- Code coverage. Program performance analysis.
- Memory contents available in real-time (Shadow RAM).
- PCMCIA compliant card for use with laptops.

MICROSOFT™
WINDOWS/'95
COMPATIBLE

To learn more, call **(408) 866-1820** for a product brochure and a FREE Demo Disk. The Demo can also be downloaded from our web site- ***http://www.nohau.com/nohau.***



For more information via your Fax, call our 24-hour Fax Center at (408) 378-2912.

nOHaU
CORPORATION

51 E. Campbell Avenue
Campbell, CA 95008-2053
Fax. (408) 378-7869
Tel. (408) 866-1820

CIRCLE # 53 ON READER SERVICE CARD

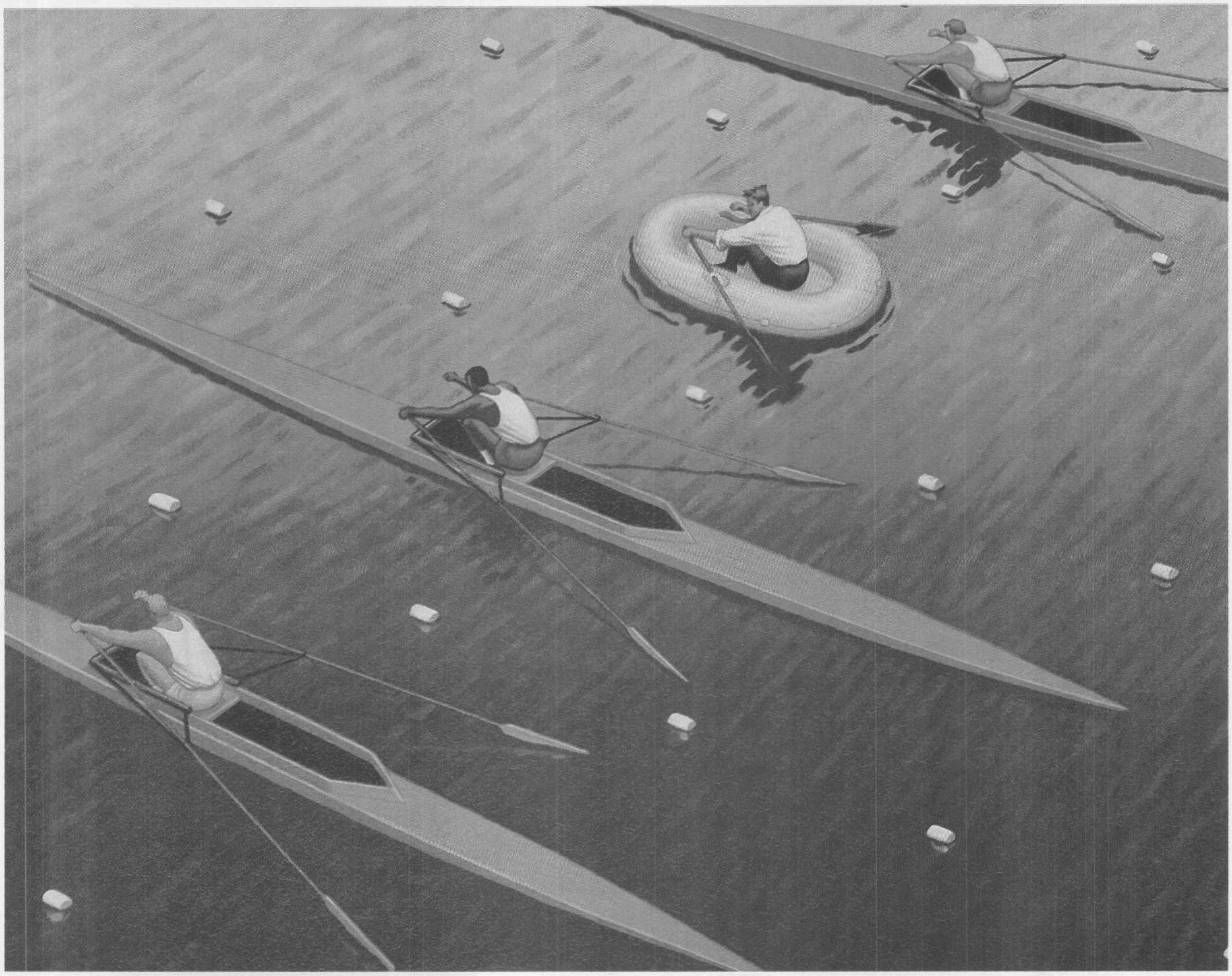
Support also available for:

Argentina 54 1 312-1079/9103, Australia (02) 654 1873, Austria 0222 27720-0, Benelux (078) 681 61 33
Brazil (011) 453-5588, Canada 514 689-5889, Czech Republic 0202-811536, Denmark 45 43 44 60 10,
Finland 90 777 571, France (1) 69 41 28 01, Germany 07043/40247, Great Britain 01962-733 140,
Greece +30-1-924 20 72, India 0212-412164, Israel 03-6491202, Italy 02 498 2051, Japan (03) 3405-0511,

NOHAU INTERNATIONAL

Korea (02) 784-7841, New Zealand 09-3092464, Norway (+47) 22 67 40 20, Portugal 01 4213141,
Romania 056 200057, Singapore +65 749-0870, S.Africa (021) 234 943, Spain (93) 291 76 33,
Sweden 040-92 24 25, Switzerland 01-745 18 18, Taiwan 02 7640215, Thailand (02) 668-5080.

8051 P51XA™ MCS®251 80C196 68HC11



It's hard to compete without the right tools.

Nobody can pinpoint problems in a Pentium® processor or Pentium® Pro system like we can.

American Arium is the undisputed leader in providing in-circuit emulators to PC manufacturers and BIOS developers. These tools and our 15+ years of ICE experience are now available to you.

Using WinDb™, our Windows™-based interface, you can:

- **View source, assembly and trace bus cycle data stored at the full 66 MHz bus speed**
- **Create multi-level complex triggers**
- **Stop the processor on memory, code and I/O accesses and other conditions**
- **Probe 16 external channels**
- **Create automated tests and eliminate redundant steps with our C-like command language**
- **Call or e-mail for free technical support**

If you can't afford to finish in second place, visit our website or call us today. Ask how your project can benefit using the right tools for the Pentium® processor and Pentium® Pro.

american
arium

14281 Chambers Road, Tustin, CA 92780
(714) 731-1661 • FAX (714) 731-6344
www.arium.com

Pentium and Pentium Pro are registered trademarks of Intel Corporation.
Windows is a trademark of Microsoft Corporation. WinDb is a trademark of American Arium.

CIRCLE # 54 ON READER SERVICE CARD

by Don Morgan

FIR Filter Design, Part 2

This month, we continue with an exploration of the basics of FIR filter design. In the last issue, we introduced response<->system response, an important Fourier transform pair impulse, and we discussed how it made the process of deriving filter coefficients easy. The Fourier transform actually provides many such aids. In this issue, we will introduce two more: the concept of odd and even and time domain convolution<->frequency domain multiplication.

Common arguments in mathematics show certain integrals vanish without the need for evaluation due to symmetry. Indeed, symmetry and periodicity provided the foundation for the Fast Fourier transform. An awareness of symmetry and its effects is valuable, as symmetry appears again and again in signal processing.

A function *E(x),* for which *E(-x) = E(x),* is a symmetrical (or even) function. Another function, *O(x),* which conforms to the equality *O(-)=-O(x),* is anti-symmetrical, or odd. (An important note is that the sum of even and odd functions is typically neither even nor odd.) Any function may be split into a sum of odd and even parts in the following manner: if $h(x)=E_1(x)+O_1(x) =E_2(x)+O_2(x)$, then E_1-E_2=O_2-O_1 with E_1-E_2 remaining even and O_2-O_1 odd, this leaves E_1-E_2 =0.

The even part of the function represents the mean of the function and its reflection on the vertical axis, while the odd part is the mean and its negative reflection. This leads us to a very useful pair of equalities:

$$E(x) = \frac{1}{2}\left[h(x) + h(-x)\right]$$

$$O(x) = \frac{1}{2}\left[h(x) - h(-x)\right]$$

Odd length filters are popular because they produce a filter delay equal to an integer number of samples.

This dissociation into odd and even changes with changing origins; functions such as cos*x* move from fully even to fully odd by a shift of origin.

An clear example of odd/even is the formulation for the Fourier series:

$$\frac{1}{2}A_0 + \sum_{n=1}^{\infty}\left(A_n \cos(nx) + B_n \sin(nx)\right)$$

If a function is either fully odd or fully even, one of the trigonometric functions need not be evaluated. For example, let us say that we wish to represent the function *f(x)=x* in the interval (0, 2π) as a Fourier series, as shown in Figure one:

$$A_0 = \frac{1}{\pi}\int_0^{2\pi} x\,dx = 2\pi$$

$$A_n = \frac{1}{\pi}\int_0^{2\pi} x\cos(nx)\,dx = 0,\ n = 1,2,3,\ldots$$

$$B_n = \frac{1}{\pi}\int_0^{2\pi} x\sin(nx)\,dx = -\frac{2}{n},\ n = 1,2,3,\ldots$$

The even function evaluates to zero and may be omitted from the series.

ODD AND EVEN FILTERS

Last month I presented an algorithm for deriving odd length FIR filters, but FIR filters may be either even or odd. Odd length filters are popular because they produce a filter delay equal to an integer number of samples. If you are attempting to do some background processing, you can determine a precise integer number of samples you have to accomplish it in. There are times when some operations are shorter than others and further processing must be delayed until other actions are complete, as may be the case with very long FIR filters. A predictable integer delay count makes this very easy.

Let α equal the filter delay and *N* represent the actual number of coefficients. For these, two relationships exist:

$$\alpha = \frac{(N-1)}{2}$$

$$h(n) = h(N-1-n),\ 0 \le n \le N-1$$

If we assume that a filter with *N*=11, then α=5. The filter is symmetrical around the fifth sample. By the same token, a filter with an even length, say *N*=10 will have α=4.5 and a filter delay of 4.5 samples. Here the center of symmetry is between two samples. Figure 2A illustrates an odd length filter and Figure 2B is an even length filter.

So what? Well, the definition of a linear phase filter requires that the filter have both constant group delay and constant phase delay. Group delay is mathematically defined as the derivative of phase with respect to frequency, but it can be intuitively understood as the time required for the input to propagate to the output. Phase delay is simply phase divided by frequency.

FIGURE 1
f(x)=x in the interval (0, 2π).

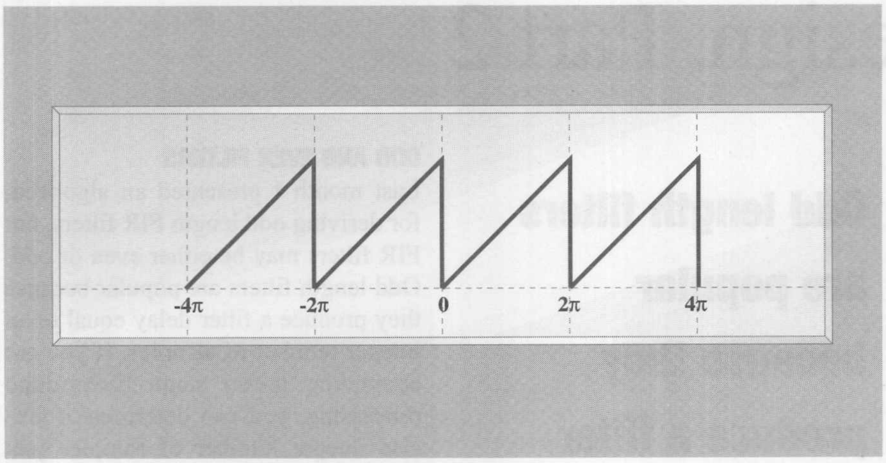


FIGURE 2A
Odd length filter.

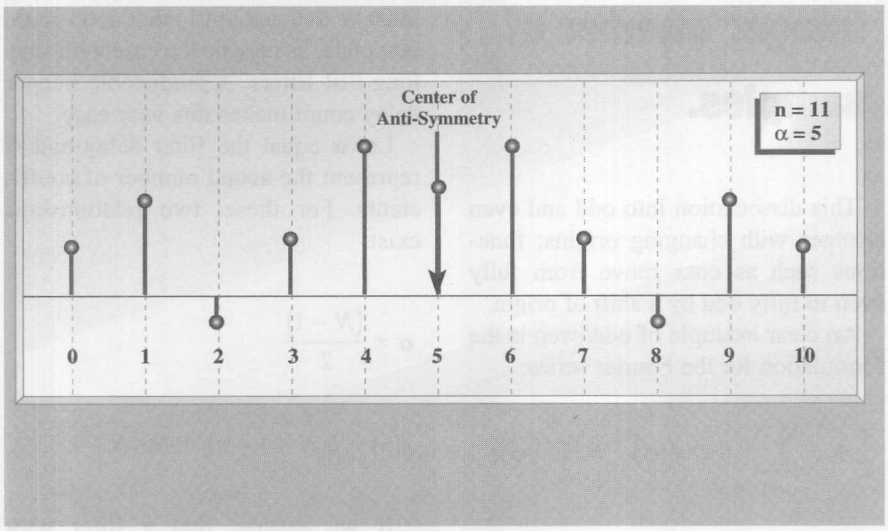
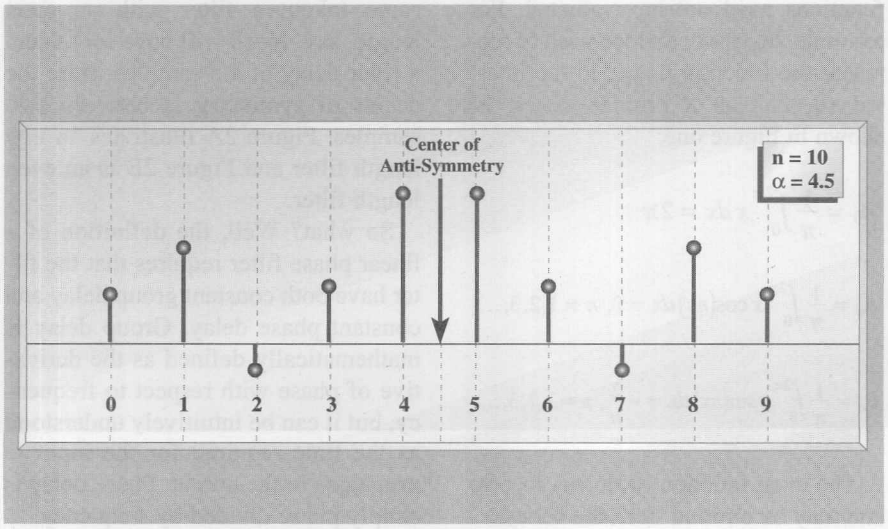


FIGURE 2B
Even length filter.



To see how symmetry and anti-symmetry are involved in filters, we can divide a system response into two parts:

$$H(e^{j\omega}) = \overline{H(e^{j\omega})}e^{j(\beta-\alpha\omega)}$$

with $H(e^{j\omega})$ representing magnitude response and $e^{j(\beta-\alpha\omega)}$ representing phase, $\beta=\pm\pi/_2$. (Notice the formula for a straight line in the exponent of the phase response: β-$\alpha\omega$.) If β=0, the phase response passes through the origin and the filter will have constant group delay and linear phase. Otherwise β acts as an offset and the straight line does not pass through the origin. This means that $\Theta(\omega_1+\omega_2)\neq\Theta(\omega_1)+\Theta(\omega_2)$, making the phase not truly linear. Filters that satisfy this description are anti-symmetric. Figure 3A illustrates anti-symmetric odd length and Figure 3B illustrates an anti-symmetric even length filter.

This situation results in four possibilities for FIR filters and four closed forms for deriving the coefficients. Here are the four cases and their expressions for $\overline{H(e^{j\omega})}$(refer to):

Case 1: *Symmetrical Impulse Response, N odd*

$$H(e^{j\omega}) = \sum^{(N-1)/2} a(n)\cos(\omega n)$$

$$a(0) = h\left[\frac{N-1}{2}\right]$$

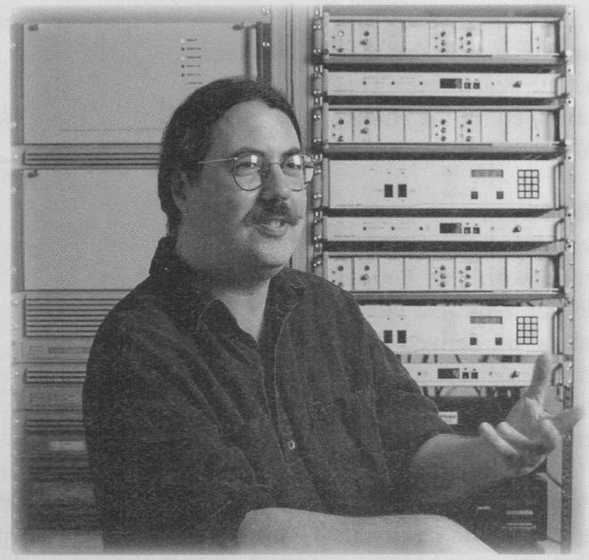
$$a(n) = 2h\left[\frac{N-1}{2} - n\right],\ n = 1,2,\ldots,\frac{N-1}{2}$$

Case 2: *Symmetrical Impulse Response, N even*

$$H(e^{j\omega}) = \sum_{n=1}^{N/2} b(n)\cos\left(\omega\left(n-\frac{1}{2}\right)\right)$$

$$b(n) = 2h\left[\frac{N}{2} - n\right],\ n = 1,2,\ldots,\frac{N}{2}$$

Case 3: *Anti-symmetrical Impulse Response, N odd*



"The Diab Data compiler is simply the best embedded compiler around. We did an evaluation of various C++ compilers for the PowerPC, and the Diab Data compiler produced the tightest, fastest code. In the last several years, I've occasionally re-evaluated the Diab Data compiler. It's still the best. Diab Data definitely lives up to its reputation for technical excellence."

Jim Houha, Kernel Engineer
PowerTV, Inc.
Set-Top Box Innovator

The Expert's Choice!

Diab Data's highly optimizing C & C++ compiler suites are the expert's choice for advanced embedded applications. When you need the fastest, tightest, most reliable code possible, Diab Data's Power Compiling Solutions™ deliver.

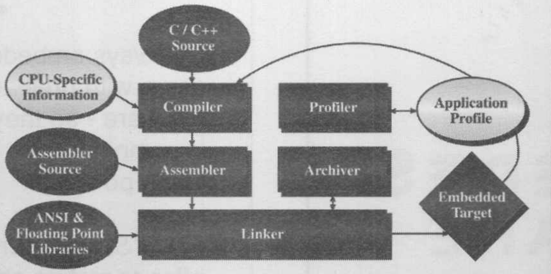
The Performance Leader

Diab Data's D-CC and D-C++ compiler suites set the performance standard for fast, compact, high-quality code. With both CPU-specific as well as application specific optimizations, D-CC and D-C++ generate the optimum code for enabling faster, more competitive applications.

Superior Programming Flexibility

Diab Data's Power Compiling Solutions offer superior programming flexibility. With precise control over compiler, optimization and linker options, you can tailor programs for the extreme design constraints of today's high-performance embedded designs.

CPU and Application Specific Optimizations Generate the Optimum Code for Your Design



Open Solutions

Diab Data's standards compliance and open environment philosophy allow you to choose from an unparalleled selection of debug and run-time environments with proven tool interoperability.

The Motorola RISC Leader

Diab Data is the market leader for embedded PowerPC and ColdFire applications.

Come join the growing number of expert developers who enjoy the superior performance, reliability and responsive service provided by Diab Data.

Diab Data's Power Compiling Solutions. The Expert's Choice. For Experts Like You.

MOTOROLA
EMBEDDED
PLATINUM DEVELOPER

Browse Us
www.ddi.com

E-mail Us
info@ddi.com

DIAB DATA
Defining Compiler Performance

CIRCLE # 55 ON READER SERVICE CARD

Diab Data Headquarters: Tel: 415-571-1700, Fax: 415-571-9068. Central Region: Tel: 630-529-6950. Eastern Region: Tel: 215-362-1786.
EUROPE: Diab Data GmbH, Tel: +49 (0)89-9393-1191, Fax: +49 (0)89-930-5184

$$\overline{H(e^{j\omega})} = \sum_{n=1}^{(N-1)/2} c(n)\sin(\omega n)$$

$$a(n) = 2h\left[\frac{N-1}{2} - n\right],\; n = 1,2,\ldots,\frac{N-1}{2}$$

Case 4: *Anti-symmetrical Impulse Response, N even*

$$\overline{H(e^{j\omega})} = \sum_{n=1}^{N/2} d(n)\sin\left(\omega\left(n-\frac{1}{2}\right)\right)$$

$$d(n) = 2h\left[\frac{N}{2} - n\right],\; n = 1,2,\ldots,\frac{N}{2}$$

Besides the need for an integer group delay, the choice between odd and even is usually most critical when one form will result in a filter whose midpoint frequency falls as close as possible to the required transition frequency of the filter. Additionally, Cases 3 and 4 have an advantage in some kinds of applications in that they contain a constant $\pi/2$ offset. This makes them well suited for the Hilbert transform or differentiation.

LAGRANGE FILTERS

Even though we have been spending most of our time with the Fourier series, please do not think that all filters must somehow be derived from the Fourier series. Many possible closed form structures are available for the development of the FIR filter. Actually, any reliable polynomial approximation technique can be made to work—examples include (we mention only a few!) the Newton interpolation formula, as well as Hermite and Taylor forms. At this time, the best known and most clearly understood technique is probably the Lagrange interpolation method.

The system function for the Lagrange interpolation method may be written:

$$H(z) = \prod_{n=0}^{N-1}\left(1 - z^{-1}z_n\right)\sum_{m=0}^{N-1}\frac{A_m}{1 - z^{-1}z_m}$$

in which:

$$A_m = \frac{H(z_m)}{\prod\limits_{\substack{n=0\\ n\neq m}}^{N-1}\left(1 - z_n z_m^{-1}\right)}$$

In its rawest form, the method produces a filter that has both poles and zeros. The Lagrange interpolation formula results in a cascade of N first-order sections containing zeros at $z=z_n$, $n=0, 1 \ldots N-1$ in cascade, with a parallel combination of N first-order sections with poles at $z=z_n$, $n=0, 1\ldots N-1$. The pole of each parallel path exactly

THE **LEADER** IN EMBEDDED INTERNET PROTOCOLS

TCP/IP

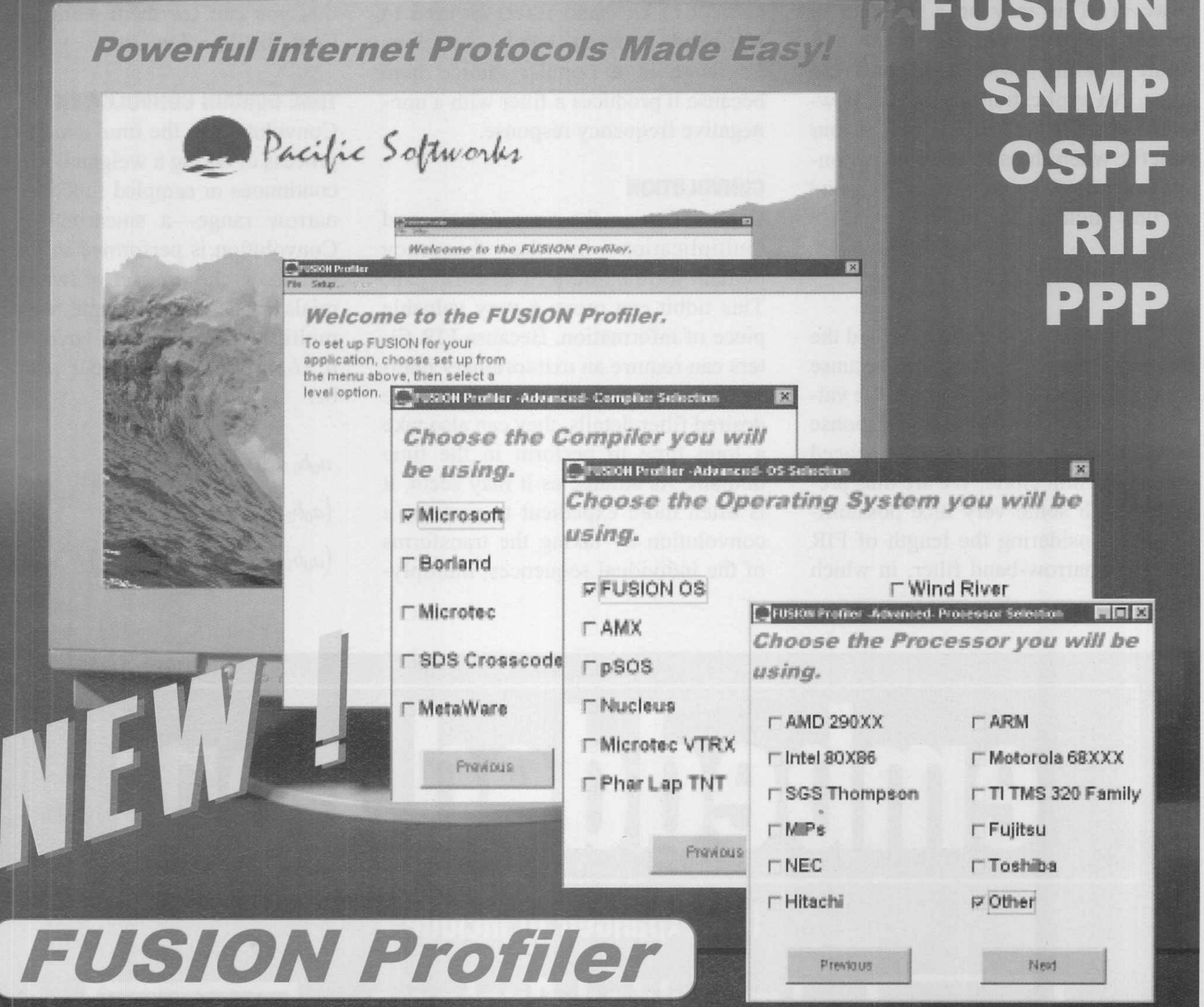
100% RFC Compliant

Web Management
SURF360/386

in **FUSION**

SNMP
OSPF
RIP
PPP

Powerful internet Protocols Made Easy!



FUSION Profiler

Welcome to the FUSION Profiler.

FUSION Profiler

File Setup

Welcome to the FUSION Profiler.

To set up FUSION for your application, choose set up from the menu above, then select a level option.

FUSION Profiler -Advanced- Compiler Selection

Choose the Compiler you will be using.

☑ Microsoft
□ Borland
□ Microtec
□ SDS Crosscode
□ MetaWare

Previous

FUSION Profiler -Advanced- OS Selection

Choose the Operating System you will be using.

☑ FUSION OS
□ AMX
□ pSOS
□ Nucleus
□ Microtec VTRX
□ Phar Lap TNT
□ Wind River

Previous

FUSION Profiler -Advanced- Processor Selection

Choose the Processor you will be using.

□ AMD 290XX
□ Intel 80X86
□ SGS Thompson
□ MIPs
□ NEC
□ Hitachi
□ ARM
□ Motorola 68XXX
□ TI TMS 320 Family
□ Fujitsu
□ Toshiba
☑ Other

Previous Next

FUSION Profiler

FUSION, the most powerful Internet protocols speeds your time-to-market and saves you $$$$$ in development costs!

FUSION Profiler gives you point-and-click control over your TCP/IP development for Embedded systems. Just select your environment, and FUSION Profiler walks you step-by-step through your configuration. This means: ease of configuration, and a perfect build every time.
Using the FUSION Profiler enables almost anyone to configure these powerful FUSION applications for embedded systems.

Pacific Softworks

FOR MORE INFORMATION CALL:
(800) 541-9508 within USA
(805) 484-2128 outside USA
(805) 484-3929 FAX
Internet: sales@pacificsw.com
Europe, Middle East and Africa
Ph: (44) 1494-432-735
FAX:(44) 1494-432-728
World Wide Web: http://
www.pacificsw.com
Distributors Worldwide

CIRCLE # 58 ON READER SERVICE CARD

cancels one of the zeros in the cascade, resulting in an equivalent filter with N-1 zeros.

The effects of finite arithmetic rarely result in complete cancellation, however. This may be a problem in certain circumstances, because it will result in a filter with both poles and zeros. An application does exist, however, where all of this works to our benefit: when the sequence of z_n consists of points equally spaced around the unit circle on the z plane:

$$z_n=e^{j(2\pi/N)n},\ n=0,1,...N\text{-}1$$

This method is sometimes called the *frequency sampling* structure, because the coefficients of the filter are the values of the filter's frequency response sampled at N points, equally spaced around the unit circle. We are thus presented with some very nice possibilities for considering the length of FIR filters. A narrow-band filter, in which only a small number of coefficients are non-zero, will only require a few multiplications per output sample.

Another popular application of Lagrange interpolation is in the development of halfband filters as used in sub-band coding and wavelets. Lagrange is a popular choice here because it produces a filter with a non-negative frequency response.

CONVOLUTION

Convolution in the time domain and multiplication in the frequency domain form a Fourier transform pair. This tidbit can prove a very valuable piece of information. Because FIR filters can require an extraordinary number of coefficients to obtain the desired filter details, they can also take a long time to perform in the time domain. As strange as it may seem, it is often more expedient to perform a convolution by taking the transforms of the individual sequences, multiplying them in the frequency domain, and then transforming the result back to the time domain.

In this section, we will look at both options and provide code for them so that you can see them work yourself. First, the time domain.

TIME DOMAIN CONVOLUTION

Convolution in the time domain is the process of taking a weighted mean of a continuous or sampled function over a narrow range—a smearing of data. Convolution is performed in the same manner as the product of two polynomials is taken. For example, we wish to multiply $a_o+a_1x+a_2x^2+a_3x^3+...$ by $b_o+b_1x+b_2x^2+b_3x^3+...$, our result will be:

$$a_0b_0+\left(a_0b_1+a_1b_0\right)x+$$
$$\left(a_0b_2+a_1b_1+a_2b_0\right)x^2+$$
$$\left(a_0b_3+a_1b_2+a_2b_1+a_3b_0\right)x^3+\ldots$$

FIGURE 3A
Anti-symmetric odd length.

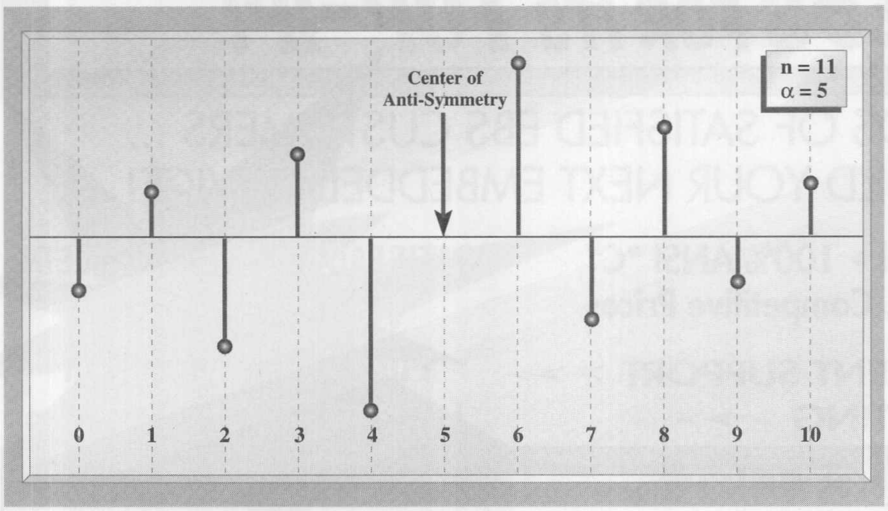


FIGURE 3B
Anti-symmetric even length.

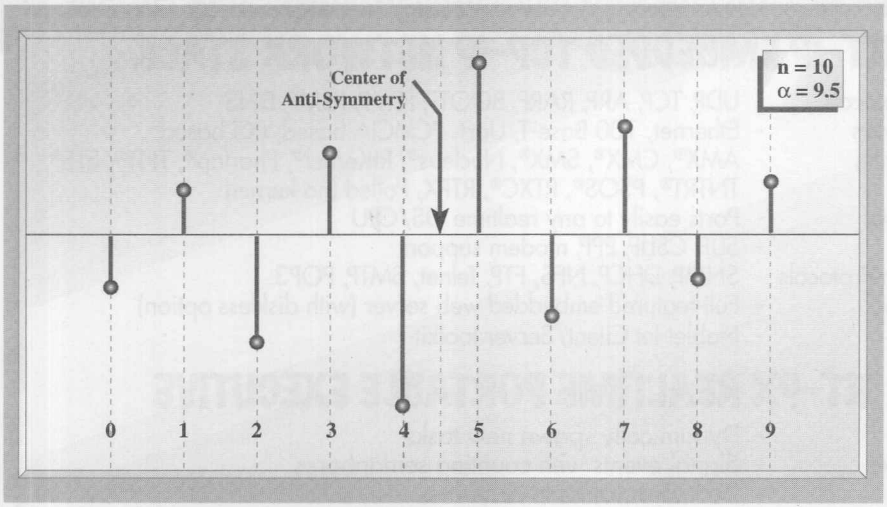
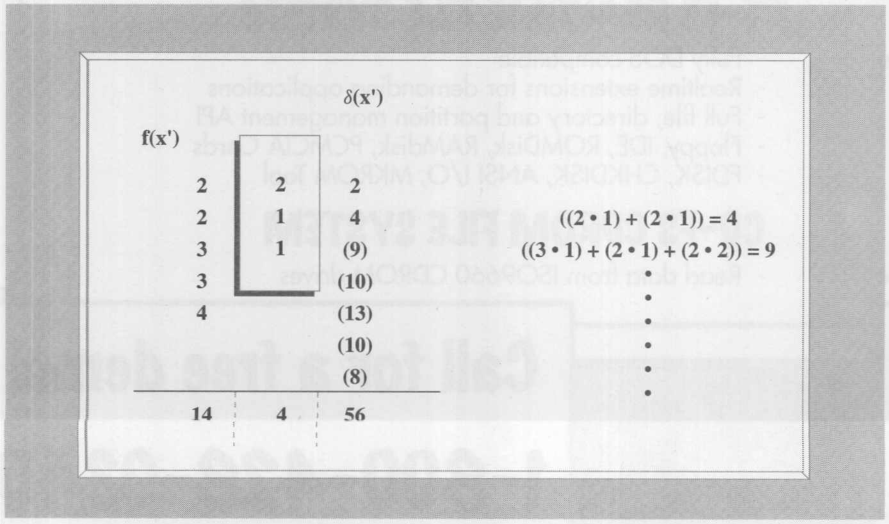


FIGURE 4
Illustration of manual numerical convolution.



We could call one polynomial $f(x')$ and the other $g(x')$ and perform the same action, but first let's refer to the convolution integral:

$$\int_{-\infty}^{\infty} f(x')g(x-x')\,dx'$$

where we note that the formula reverses the sequence $g(x')$.

The process of a numerical convolution can be demonstrated quite easily with two pieces of paper. The $g(x')$ sequence is written on a moveable piece of paper, allowing it slide alongside the values of the $f(x')$ sequence. The two sequences are written in vertical columns and the answers are written opposite an arrow marked in a convenient place on the movable strip. Figure four is an illustration of this technique.

As you see, the result of the operation is a longer sequence than the two original sequences. Actually it is one less than the sum of the numbers of terms in the two individual sequences. Of course, the process is written in sampled data form as:

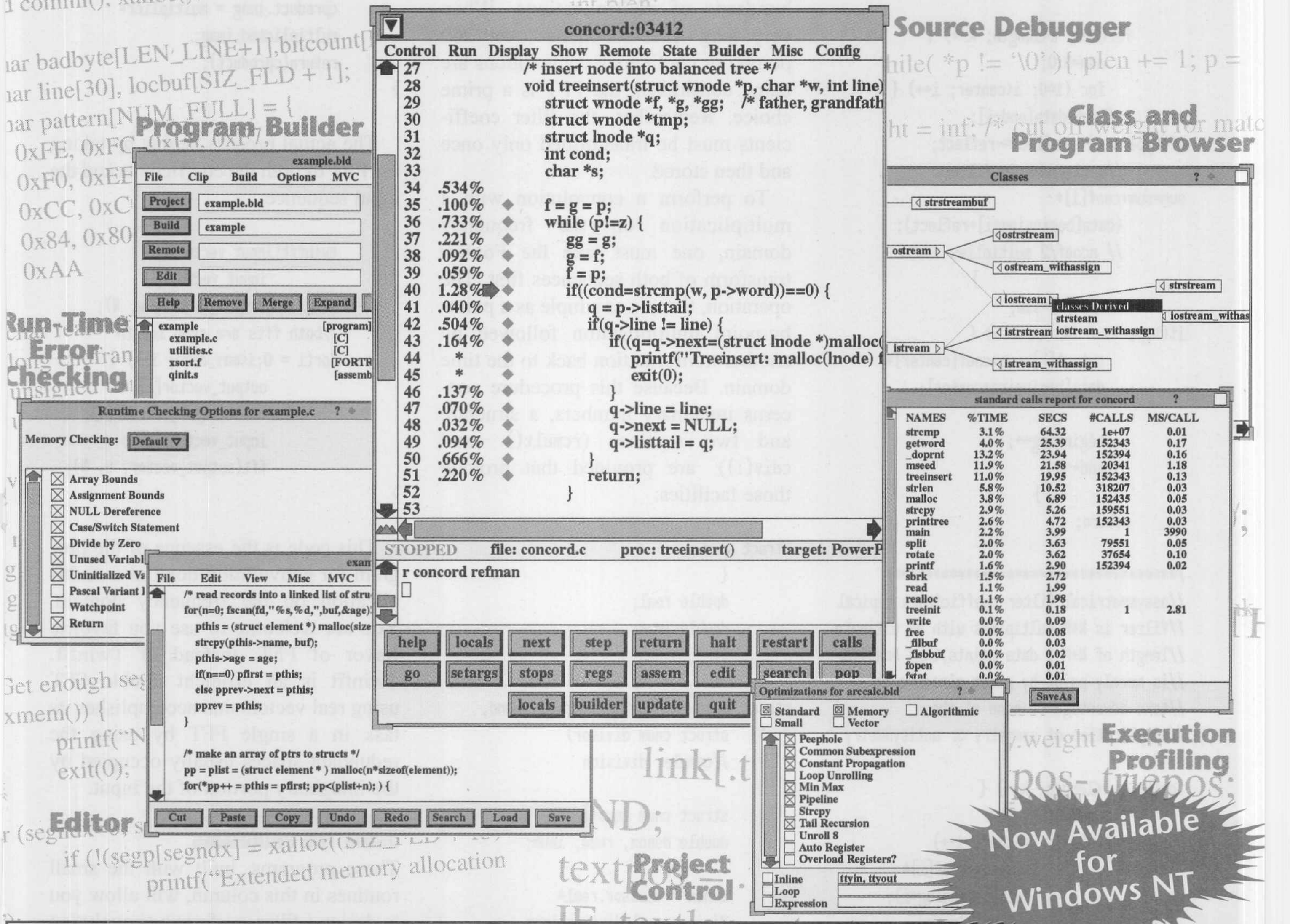
$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

This concept can be expressed in C as in the following:

```
void tconvolve(double coef[],
      int ncoef, double data[], int nn,
      double output[],
      int type, int symmetry)
{
      int center, odd, k, i, end, length,
            beginning;
      double sum, reflect;

      length = nn+1;
      beginning = 0;
    end=ncoef-1;
/*********************************/
          //symmetrical filter coefficients
    if(symmetry) {
          //exploit the symmetry in the
          //filter response

        center=ncoef>>1;
        odd=1;
```

MULTI®

The Most Advanced Software Development Environment for Embedded Applications



```
27          /* insert node into balanced tree */
28          void treeinsert(struct wnode *p, char *w, int line)
29              struct wnode *f, *g, *gg;    /* father, grandfath
30              struct wnode *tmp;
31              struct lnode *q;
32              int cond;
33              char *s;
34  .534%
35  .100%  ◆    f = g = p;
36  .733%  ◆    while (p!=z) {
37  .221%  ◆        gg = g;
38  .092%  ◆        g = f;
39  .059%  ◆        f = p;
40  1.28%  ◆        if((cond=strcmp(w, p->word))==0) {
41  .040%  ◆            q = p->listtail;
42  .504%  ◆            if(q->line != line) {
43  .164%  ◆                if((q=q->next=(struct lnode *)malloc(
44    *    ◆                    printf("Treeinsert: malloc(lnode) f
45    *    ◆                    exit(0);
46  .137%  ◆                }
47  .070%  ◆                q->line = line;
48  .032%  ◆                q->next = NULL;
49  .094%  ◆                q->listtail = q;
50  .666%  ◆            }
51  .220%  ◆            return;
52                  }
53
STOPPED     file: concord.c     proc: treeinsert()     target: PowerP
r concord refman
```

Project: example.bld
Build: example

```
example          [program]
  example.c      [C]
  utilities.c    [C]
  xsort.f        [FORTR
  qinit.s        [assemb
```

Runtime Checking Options for example.c

Memory Checking: Default

- ☒ Array Bounds
- ☒ Assignment Bounds
- ☒ NULL Dereference
- ☒ Case/Switch Statement
- ☒ Divide by Zero
- ☒ Unused Variabl
- ☒ Uninitialized Va
- ☐ Pascal Variant l
- ☐ Watchpoint
- ☒ Return

```
/* read records into a linked list of stru
for(n=0; fscan(fd,"%s,%d,",buf,&age)
   pthis = (struct element *) malloc(size
   strcpy(pthis->name, buf);
   pthis->age = age;
   if(n==0) pfirst = pthis;
   else pprev->next = pthis;
   pprev = pthis;
}

/* make an array of ptrs to structs */
pp = plist = (struct element * ) malloc(n*sizeof(element));
for(*pp++ = pthis = pfirst; pp<(plist+n); ) {
```

standard calls report for concord

NAMES	%TIME	SECS	#CALLS	MS/CALL
strcmp	3.1%	64.32	1e+07	0.01
getword	4.0%	25.39	152343	0.17
_doprnt	13.2%	23.94	152394	0.16
mseed	11.9%	21.58	20341	1.18
treeinsert	11.0%	19.95	152343	0.13
strlen	5.8%	10.52	318207	0.03
malloc	3.8%	6.89	152435	0.05
strcpy	2.9%	5.26	159551	0.03
printtree	2.6%	4.72	152343	0.03
main	2.2%	3.99	1	3990
split	2.0%	3.63	79551	0.05
rotate	2.0%	3.62	37654	0.10
printf	1.6%	2.90	152394	0.02
sbrk	1.5%	2.72		
read	1.1%	1.99		
realloc	1.1%	1.98		
treeinit	0.1%	0.18	1	2.81
write	0.0%	0.09		
free	0.0%	0.08		
_filbuf	0.0%	0.03		
_flsbbuf	0.0%	0.02		
fopen	0.0%	0.01		
fstat	0.0%	0.01		

Optimizations for arccalc.bld

☒ Standard ☒ Memory ☐ Algorithmic
☐ Small ☐ Vector

- ☒ Peephole
- ☒ Common Subexpression
- ☒ Constant Propagation
- ☐ Loop Unrolling
- ☒ Min Max
- ☒ Pipeline
- ☐ Strcpy
- ☒ Tail Recursion
- ☐ Unroll 8
- ☐ Auto Register
- ☐ Overload Registers?

☐ Inline: ttyin, ttyout
☐ Loop
☐ Expression

Software Development Tools for Embedded Development

Green Hills
• SOFTWARE, INC. •

High Performance Optimizing Compilers

C   C++   Fortran   Pascal   Ada

The MULTI® Development Environment

Source Debugger   Profiler   Run-Time Error Checking
Version Control   Editor   Source and Class Browser
Program Builder   Project Control

Cross Development Tool Chain

Assembler   Librarian   Run-Time Libraries
Linker   Utilities   Object Code Converters

Host Platforms

PC/Windows   Sun/SunOS   HP/HPUX   DEC/Alpha AXP
PC/NT   Sun/Solaris   IBM/AIX   SGI/Irix

Processor Families Supported

680x0/683x0   PowerPC   Alpha   V800
R3000/R4000   386/486   SPARC   SH
88100/88110   i960   FR20   M16
RAD6000   RH32   ARM   more...

Real Time Operating System Supported

INTEGRITY   velOSity   VxWorks
pSOS+   VMEexec   Nucleus

Target Systems

Simulator   ROM Monitor   CPU Boards   Emulators

Green Hills Software, Inc.
30 West Sola Street
Santa Barbara, CA 93101
Email: sales@ghs.com
http://www.ghs.com

Tel. (800) 856-0720
Fax. (805) 965-6343

CIRCLE # 60 ON READER SERVICE CARD

```
        if((center<<1) == ncoef)
        odd=0;

    for(k=0; k<length; k++) {
        sum=0.0;
        for (i=0; i<center; i++) {
    reflect=data[end-i];
if(type < 0)  reflect=-reflect;
        //antisymmetric filters
sum=sum+coef[i]*
      (data[beginning+i]+reflect);
      // ncoef/2 multiplies
                    }
      output[k]=sum;
if(type >= 0 && odd > 0) {
        output[k]=sum+coef[center]*
        data[beginning+center];
                    }
        beginning++;
        end++;
              }
    return;
}
/*********************************/
//assymmetrical filter coefficients typical
//filter is N*M multiplies with an ultimate
//length of N+M-1 data points,this handling
//is merely point by point since we can
//take advantage of none of the
//simplicities of symmetry or antisymmetry

for(k=0; k<length; k++) {
      sum=0.0;
      for(i=0; i<ncoef; i++)
               sum=sum+coef[i]*
               data[beginning+i];
               output[k]=sum;
               // ncoef multiplies
               beginning++;
      }
}
```

The complete program (and a small utility that allows you to display on your screen plots generated here) may be found on the *ESP* Web site at **www.embedded.com/code.htm**.

FREQUENCY DOMAIN CONVOLUTION

As mentioned earlier, it can sometimes be more expedient to perform a convolution by transforming to the frequency domain than trying to do it in the time domain. This approach may seem a bit unreal considering the computational burden of performing three FFTs, but very efficient FIR filters can require hundreds of coefficients, and therefore hundreds of multiplications. When very long filter functions (over 200 points) or continuous convolutions are being considered, the FFT is a prime choice. Remember, the filter coefficients must be transformed only once and then stored.

To perform a convolution with a multiplication on the frequency domain, one must take the Fourier transform of both sequences first. The operation, then, is as simple as a point-by-point multiplication followed by another tranformation back to the time domain. Because this procedure concerns imaginary numbers, a structure and two routines (rcmult() and cdiv(:)) are provided that provide those facilities:

```
struct cnum
       {
       double real;
       double imag;
       };


struct cnum cdiv(struct cnum dividend,
       struct cnum divisor)
       //complex division
       {

       struct cnum cquotient;
       double denom, rnum, inum;

       denom = divisor.real*
       divisor.real+divisor.imag
       *divisor.imag;
       rnum = dividend.real*
       divisor.real +
       dividend.imag*
       divisor.imag;
       inum = dividend.imag*
       divisor.real -
        dividend.real*
       divisor.imag;
       cquotient.real = rnum/denom;
       cquotient.imag = inum/denom;
       return(cquotient);
}


struct cnum rcmult(double multiplier,
       struct cnum multiplicand)
       //real by complex multiplication
       {

       struct cnum cproduct;
       cproduct.real = multiplier*
       multiplicand.real;
       cproduct.imag = multiplier*
       multiplicand.imag;
       return(cproduct);
}
```

The actual process begins by taking the FFT of both the coefficients and the input sequence:

```
       twinfft(input_vector0,
               input_vector1,
               output_vector, w, N);
       //both ffts are same length
       for(i = 0;i<max_data; i++) {
               output_vector[i] =
               cmult(input_vector0[i],
               input_vector1[i]);
               fft(output_vector, w, N);
}
```

This code is the essence of the program for convolution done as a multiplication on the frequency domain. You are welcome to use you favorite flavor of FFT, instead of Twinfft. Twinfft is an efficient double FFT using real vectors that accomplishes its task in a single FFT by using the redundant spaces usually occupied by the imaginary portion of the input.

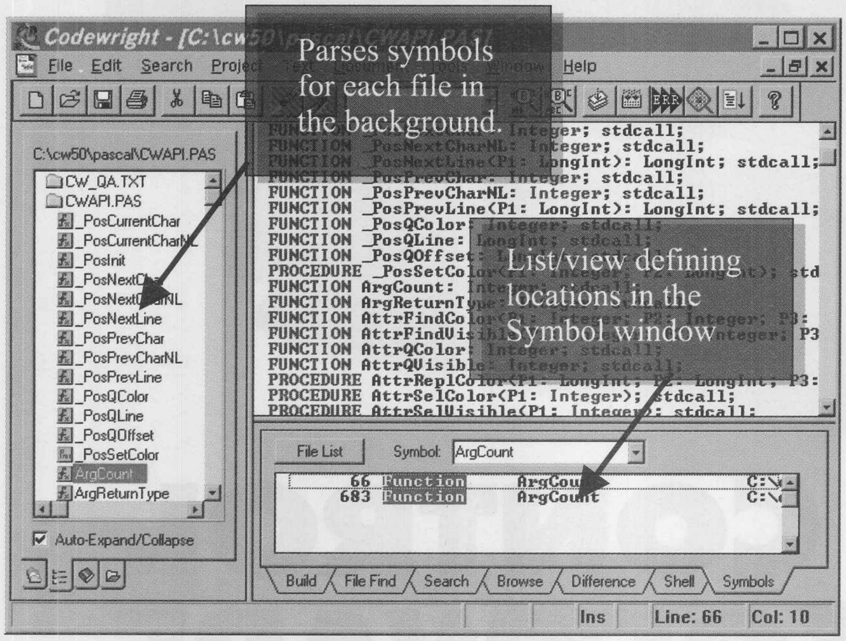
USING THE PROGRAMS

These programs, built with the small routines in this column, will allow you to design a filter, perform a convolution with a data vector, and see the results. Each program expects coefficients and a data vector which may be generated by hand or by many common programs, such as Matlab, Mathcad, Excel, and so on—the sequence must simply be space delimited.

I hope that these programs clarify and remove some of the mysticism involved in filter design. They are obviously not complete or perfect but they do embody the concepts used in such designs. ESP

Don Morgan is a senior engineer at Ultra Stereo Labs and a consultant in signal processing, embedded systems, hardware, and software. He's completing a book about numerical methods.

Raise the bar
by lowering your hurdles.



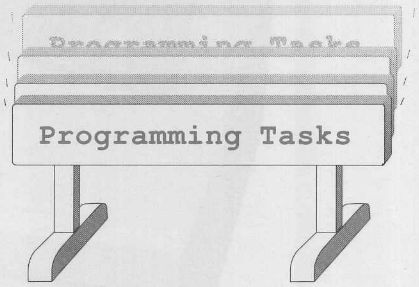
```
FUNCTION _PosNextChar: Integer; stdcall;
FUNCTION _PosNextCharNL: Integer; stdcall;
FUNCTION _PosNextLine(P1: LongInt): LongInt; stdcall;
FUNCTION _PosPrevChar: Integer; stdcall;
FUNCTION _PosPrevCharNL: Integer; stdcall;
FUNCTION _PosPrevLine(P1: LongInt): LongInt; stdcall;
FUNCTION _PosQColor: Integer; stdcall;
FUNCTION _PosQLine: LongInt; stdcall;
FUNCTION _PosQOffset: LongInt; stdcall;
PROCEDURE _PosSetColor(P1: Integer; P2: LongInt); std
FUNCTION ArgCount: Integer; stdcall;
FUNCTION ArgReturnType: Integer; stdcall;
FUNCTION AttrFindColor(P1: Integer; P2: Integer; P3:
FUNCTION AttrFindVisible(P1: Integer; P2: Integer; P3
FUNCTION AttrQColor: Integer; stdcall;
FUNCTION AttrQVisible: Integer; stdcall;
PROCEDURE AttrReplColor(P1: LongInt; P2: LongInt; P3:
PROCEDURE AttrSelColor(P1: Integer); stdcall;
PROCEDURE AttrSelVisible(P1: Integer); stdcall;
```

```
       66 Function      ArgCount          C:\
      683 Function      ArgCount          C:\
```

Parses symbols for each file in the background.

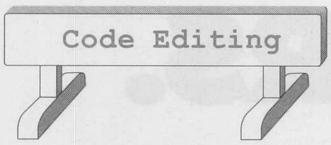
List/view defining locations in the Symbol window

Navigate code *fast* with Outline Symbols



With a Codewright™ V5.0 Professional Programmer's Editor

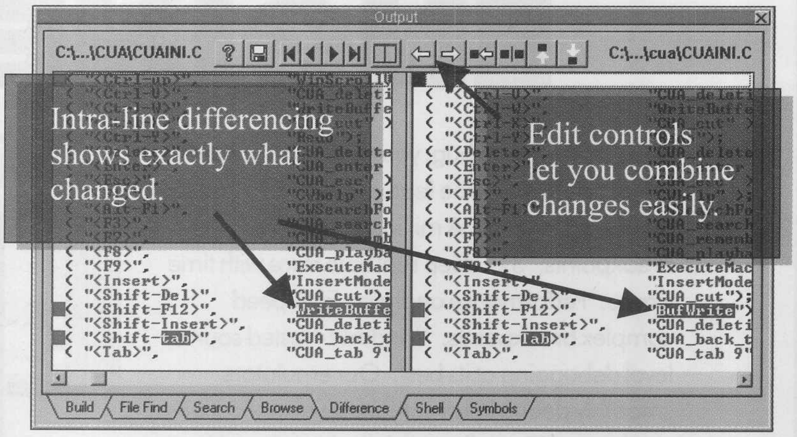
from premia®
Premium Quality Software



Codewright helps you over tasks like:

- Locating and navigating to specific code.
- Creating and maintaining comments.
- Working with version control.
- Typing repetitive constructs.
- Renaming classes and variables across hundreds of files.
- Focusing on what's important.
- Working with others *and* their revisions.
- Linking to non-text files.
- Your next raise (maybe).

Codewright synchronizes with your Visual C++ and Delphi environments, and offers the best free support in the business. Call about a free evaluation.



```
( "<Ctrl-Up>",        "WinScrollU       ( "<Ctrl-V>",          "CUA_deleti
( "<Ctrl-V>",         "CUA_deleti       ( "<Ctrl-W>",          "WriteBuffe
( "<Ctrl-W>",         "WriteBuffe       ( "<Ctrl-X>",          "CUA_cut" >
( "<Ctrl-X>",         "CUA_cut" >       ( "<Ctrl-Y>",          "Redo");
( "<Ctrl-Y>",         "Redo");          ( "<Delete>",          "CUA_delete
( "<Delete>",         "CUA_delete       ( "<Enter>",           "CUA_enter
( "<Enter>",          "CUA_enter        ( "<Esc>",             "CUA_esc" >
( "<Esc>",            "CUA_esc" >       ( "<F1>",              "CWhelp" >;
( "<F1>",             "CWhelp" >;       ( "<Alt-F1>",          "CWSearchFo
( "<Alt-F1>",         "CWSearchFo       ( "<F3>",              "CUA_search
( "<F3>",             "CUA_search       ( "<F7>",              "CUA_rememb
( "<F7>",             "CUA_rememb       ( "<F8>",              "CUA_playba
( "<F8>",             "CUA_playba       ( "<F9>",              "ExecuteMac
( "<F9>",             "ExecuteMac       ( "<Insert>",          "InsertMode
( "<Insert>",         "InsertMode       ( "<Shift-Del>",       "CUA_cut");
( "<Shift-Del>",      "CUA_cut");       ( "<Shift-F12>",       "BufWrite">
( "<Shift-F12>",      "WriteBuffe       ( "<Shift-Insert>",    "CUA_deleti
( "<Shift-Insert>",   "CUA_deleti       ( "<Shift-Tab>",       "CUA_back_t
( "<Shift-tab>",      "CUA_back_t       ( "<Tab>",             "CUA_tab 9"
( "<Tab>",            "CUA_tab 9"
```

Intra-line differencing shows exactly what changed.

Edit controls let you combine changes easily.

Reconcile changes with Difference Editing

It's **Tool One.**

Call now: 1-888-4-PREMIA
477-3642

http://www.premia.com

Single user license:
Codewright Professional
for Windows, Windows 95, and NT.

$269

Volume discounts and site licenses available.

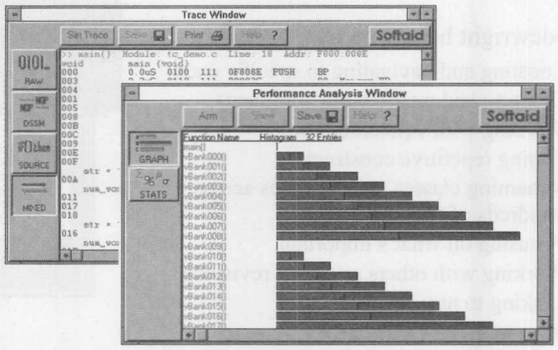
Premia Corporation
1075 NW Murray Blvd., Suite 268
Portland, Oregon 97229 USA
Fax: +1-503-641-6001
Phone: +1-503-641-6000
Email: sales@premia.com

CIRCLE # 61 ON READER SERVICE CARD

UEM Series
In circuit emulator

PEST CONTROL FOR SOFTWARE ENGINEERS.

If you can't find the bug, you can't fix it. Softaid's emulators have all the features you need to find and fix those bugs. A million hardware breakpoints. 32k deep real time trace with time stamp. Performance analysis. Full speed complex breakpoints. Windows-hosted source level debugging at its best. Our emulators smoothly debug interrupt-intensive applications: single step ISRs, collect and display trace without stopping execution, and view/set variables and breakpoints on the fly. And best of all, prices range from $5,000 to $10,000. Making the Softaid UEM your only choice for serious bug-fighting.



```
>> main(): Module: tc_demo.c  Line: 18  Addr: F800:008E
void            main (void)
000             0.0uS  0100  111  0F808E  PUSH   BP
003
004
001
005
008
00B
00C
009
00E
00F
          str =
00A
          num_vo
011
017
018
          str =
016
          num_vo
```

We emulate these processors:
80188/186, 386EX, Z180, Z80, 8088/86, V20/30, V40/50, 8085, 68000, 68HC16

softaid

Setting the standard in source level debugging

HTTP://www.softaid.net/embedded.html

8310 Guilford Rd, Columbia, MD 21046
Phone: 800-433-8812   410-290-7760
Fax: 410-381-3253

CIRCLE # 112 ON READER SERVICE CARD

BREAK POINTS

by Jack G. Ganssle

Vanishing Visibility

I take great satisfaction in my tools. My fingers are not strong enough to remove a bolt, but give me a wrench and my hand can perform amazing new feats. We computer folks like to consider the PC a mind tool that increases the power and reach of one's brain. Conventional hand tools give us a similar ability to manipulate the mechanical world in ways impossible via the unaided human body.

I'm a fanatic about woodworking tools; I keep them clean and sharp, buy only the best, collect the cream of the technology of yesteryear that, while now out of style, may still be the best solution to a problem. Though power tools with big motors that hurl sawdust like a swirling gale satisfy my testosterone-pumped craving for brute mechanical power, high quality chisels and planes are among my favorite possessions. A hand plane works well only if you take time to understand the wood, molding the plane's use to the grain, hardness, and even moisture content of the wood. In contrast, a 2hp electric plane blindly tears through any obstacle, leaving its marks of destruction behind in telltale chatter-gouges. Yet you can't beat an electric plane for removing lots of wood fast.

The same goes for the embedded world. This magazine bulges with ads for all sorts of virtual assistants, each of which is aimed at one part of the development process. Just as the hand and electric planes have valid (though different) applications, no single embedded instrument is the silver bullet for all circumstances. One of the skills of the engineer is the judicious selection and use of the right mix of tools for each project.

This fascination with tools of all kinds led me to start an emulator company back in the '80s. It has been a wild and fascinating ride, made much more interesting by the opportunity to look into the work of thousands of developers and to see how we grapple with the bugs that plague even the most well designed systems. Recently, though, I decided to move on and sell the company.

No one embedded instrument is the silver bullet for all circumstances. One key skill is the judicious selection and use of the right mix of tools.

Yet the problem of getting products to market still fascinates me. My love of tools of all sorts is undiminished. With no longer any equity in the tool business and thus no conflict of interest with this column, I feel freer to examine some of the issues that are surfacing in the '90s.

And I'm concerned. Scared, really, for the future of embedded developers. The industry is driven by relentless forces none of us can control and can sometimes barely understand. The twin forces of technology advancements and frenetic business are backing engineers into a metaphorical corner of impossible demands with terribly limited resources.

Now systems are more complex than ever, with new breeds of bugs. Timing problems, once restricted to hardware, are an ever more problematic firmware fact. RTOS complexities and excruciatingly complex algorithms fan the fire of bugs.

Bugs will never go away. Better development methodologies can reduce the error rate, but never to zero, and certainly not until we individual developers create a personal passion for improvement. Debuggers—of many types—will always be important tools.

Debuggers do one fundamental thing: provide visibility into your system. Features vary, but all we ask of a debugger is "tell me what is going on!" Sometimes we're interested in procedural flow (single stepping, breakpointing); other times function timing or dependencies or memory allocation. Regardless, we simply expect our tools to reveal hidden system behavior. Only after we see what's going on can we use our brains to understand "why that happened," and then apply a fix. My fear is that we're removing our ability to look into the systems. The visibility we take for granted is being eroded.

TECHNOLOGY TRIBULATIONS

In embedded systems, emulators have always been one of the choice weapons in the war on bugs. Yet, for as long as I can remember, pundits have been predicting their death. Though this bit of doomsaying seems as quaint as the '50s IBM prediction of the worldwide demand for computers totalling no more than a couple of dozen, in fact 20 years ago many people believed that the 4MHz Z80 would spell the doom for ICEs. "Four megahertz is just too fast," they proclaimed. "No one can run those speedy signals down a cable."

Time proved them wrong, of course. Today's units run at 60MHz-plus on processors with single-clock memory cycles, an astonishing achievement.

The imagined speed limit is not limited to ICEs, as ROM emulators and other debuggers that use a physical target system interface all suffer from similar problems. Is an end yet in sight? I believe so, though the limiting frequency is a bit hazy. Today's approach of putting all or much of the ICE's electronics on the pod removes the cabling and bus driver problems, but electrons do move at a finite speed and even the fastest of circuits have non-zero propagation delays.

CPU vendors squeeze the last bit of clock rates from their creations partly by tuning their chips ever more exquisitely to the rest of the system's memory and I/O. The current problem with PC motherboards is a danger signal: it is so difficult to design a high speed Pentium-based motherboard that Intel has had to assume that role. They are reportedly now the largest producer of PC motherboards. In effect the computer is so tightly coupled to the processor that only the CPU vendor can produce a reliable system based on the chip! Clearly, an intrusion by any sort of development tool will at best be problematic. Yes, today's Pentium emulators do work. Will tomorrow's units be able to handle the continued push into stratospheric clock rates? I have doubts.

Packages are creating another sort of problem. Heat, speed, and size constraints have yielded a proliferation of packaging styles that challenge any sort of probing for debugging. If you've ever tried to use a scope on a 208-pin PQFP device or, worse, a 100-pin TQFP, you know what I mean. Yes, some tremendously innovative probing systems exist—notably those from Emulation Technology and HP. Despite these, it's still difficult at best to establish a reliable connection between a target CPU and any sort of hardware debugger, from a voltmeter to an ICE.

Traditional surface mount devices (How can a few-year-old technology have traditions?) have exposed pins that you at least have a prayer of getting to. Newer devices don't. The BGA (Ball Grid Array) package, which is suddenly gaining favor, connects to a PC board via hundreds of little bumps on the underside of the package—where they are completely inaccessible. Other technologies bond the silicon itself directly to the board, under a dab of epoxy. All of these trends offer various system benefits; all make it difficult to impossible to troubleshoot software and hardware.

OK, you smirk, these issues only apply to the high end of the embedded market, where clock rates—and production costs—soar with the eagles. Other subtle influences, though, are wreaking havoc on the low end.

Take microcontrollers, for example. These CPUs have ROM and RAM on-board, giving a very simple, very inexpensive one-chip solution for simple 8- and 16-bit applications. The 8051 is the classic example of this; it's been an amazing success that has survived 20 years of assault by other, perhaps more capable, processors.

Single chip solutions are tough to debug, though, because the on-board memory means there's generally no address/data bus coming to the outside world. An extreme example is Microchip's 8-pin PIC part. Eight pins! The only ins and outs are I/O.

Various debugging solutions exist, but the traditional solution is the bond-out chip, a special version of the processor, with extra pins that bring all important signals to the outside world, especially those oh-so-critical address and data lines needed to track program execution. With a proper bond-out-based ICE, you can track, in real time, everything the code does with no compromises. Perfect, no?

Well, a few wrinkles are starting to surface. For one, the chip vendors *hate* making bond-outs. The market is essentially zero, yet every time the processor's mask gets revised, a new bond-out is needed. In the old days chip vendors swallowed hard, but did make them reasonably available.

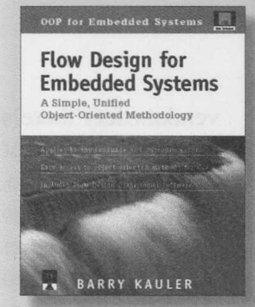
Now this is less common. With the 386EX (which is not a microcontroller, but does benefit from a bond-out) Intel announced that only a handful of vendors would get access to the special version of the part, probably increasing the cost of tools to some extent. Is this an indication of the beginning of the end of generally available bond-out parts?

Sometimes the bond-out is not kept to current mask revisions. I know of at least one case in which a vendor provides bond-outs that will not run at full speed, essentially removing the critical visibility of real-time execution from developers. This situation puts you in the awful conundrum of deciding "Should I buy an expensive tool that forces me to run at half speed, no doubt destroying all timing relationships?"

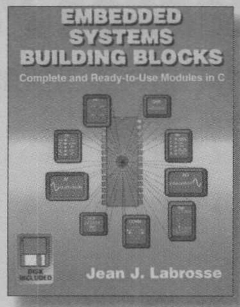
Sometimes—often—the bond-outs will not run at reduced voltages. Your 3V system might require a pod which is a convoluted mix of 3V and 5V technologies, creating additional propagation delays as voltages get translated. In effect, a non-intrusive tool becomes subtly more intrusive, in ways that are hard to predict. Voltages are declining fast—some CPUs now run at sub-1V levels—so the problem can only get worse.

A very scary development is the incredible proliferation of CPUs. Vendors are proud of their ability to crank out a new chip by pressing a few buttons on a CAD system, changing the mix of peripherals and memory, producing variant number 214 in a particular processor family. Variants are a sign of a good, healthy line of parts (look at that mind boggling array of 8051 parts), but are a nightmare for tool vendors. Each requires new hardware, software, support, evaluation boards, and the like. In the "good old days," when we saw only a few new parts per year per family, support was easy to find. Now my friends who make microcontroller tools complain of the frantic pace needed to support even a subset of the parts.

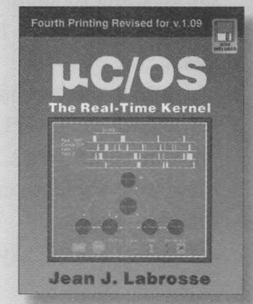
As tool consumers you probably don't care about the woes of the vendors. But part proliferation creates a problem that hits a bit closer to home: for any specific variant there may only be a handful of customers. Tool support may never exist for that part if



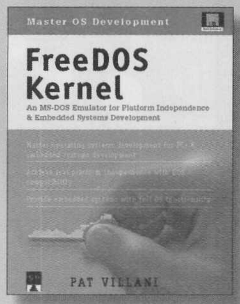
OOP for Embedded Systems
Flow Design for Embedded Systems
A Simple, Unified Object-Oriented Methodology
BARRY KAULER



EMBEDDED SYSTEMS BUILDING BLOCKS
Complete and Ready-to-Use Modules in C
Jean J. Labrosse



Fourth Printing Revised for v.1.09
µC/OS
The Real-Time Kernel
Jean J. Labrosse



Master OS Development
FreeDOS Kernel
An MS-DOS Emulator for Platform Independence & Embedded Systems Development
PAT VILLANI

vendors feel there's not a big enough market. An odd fact of the tool market (from compilers to ICEs) is that the health of the market is a function of the number of customers using a chip, not the number of chips used. CPU vendors are happy to get one or two huge design wins, say an automotive company that sucks up millions of parts per year. Tool folks might only sell a couple of units to such a customer, far too few to pay their huge development costs.

I know of dozens of big companies left stranded by CPUs with no support, who in some cases, have had to build their own tools. Some even write customer compilers! There can't be a more expensive way to do a project.

NON-COMPUTERS

As one interested in the philosophical implications of our business, I'm fascinated with the drift to "virtual" implementations of, well, everything. Hey, your cell phone is a fascinating connection—without wires—to a billion other phones on the planet; it's part of the biggest machine on the planet, yet looks like nothing more than a few bucks of electronics. Similar virtual connections underlie just about everything on the Internet as well. Now we're seeing a move to "virtual" microprocessors. Today, you can buy a micro, that, well, just has no physical being. It's a file of VHDL equations.

Buy a virtual Z80 or 186 and then incorporate that into your own design. Burn it into an FPGA or ASIC. The idea is to reduce chip count by integrating the processor into the ASIC along with all of your proprietary circuits. It keeps costs and board size down.

We're used to software being a rather ethereal "thing," with no real physical implementation. Now we can buy "hardware" equally as ethereal. It's software hardware. Hardware software. Or something.

Some of the vendors promoting these ghostly CPUs promise the ability to customize the processor. Add instructions with a click of the mouse! It would seem a magic solution to precisely match computational power to your application's needs. But how will you use the new instructions? Code in assembly language only? Write your own compiler? Worse, with the CPU buried inside of a big chip, how do you plan to troubleshoot your code?

TO BE CONTINUED

Cache, prefetchers, superscaler designs, and lots of other ever-more-common processor features all create debugging headaches. My point here is not a complaint against the technology; my point is to voice a concern that we dare not blindly design in the latest cool thing without understanding how we'll find our bugs. We've got to realize that these new features have both benefits and perils. I have seen too many designers in the flush of initial project optimism forget that soon they'll be up to their eyeballs in bugs, and that they will need some sort of tool to give them visibility into their code.

Technological problems are a funny thing. The barriers rarely stand for long. Customer needs quickly translate into solutions. One only has to look at IBM's PowerPC parts, some of which include a built-in debug port that even supports real-time trace, to see what the future might bring.

The tool vendors are incredibly innovative and will surely continue to issue a stream of wonderful inventions aimed at easing our work. But as I'll explore next month, these technology problems are now accompanied by a counterpoint of even more serious business issues. I believe that time to market, capital costs, and simple lack of knowledge causes management to make silly decisions that, coupled with the technology problems, cripple development projects. ESP

Having recently sold Softaid, Jack Ganssle is hunting for a sailboat (he promises not to sink this one) while helping companies improve their embedded engineering processes. Contact him at jack@ganssle.com.

Every day you are asked for the impossible.

Every day you give it.

education

inspiration

growth

dedication

Even superheroes need a little *inspiration.*

Embedded Systems Conference

Sept. 29 - Oct. 2, 1997 San Jose, CA

Come to the technical conference for embedded design superheroes.

Four days of new ideas, new products, practical skills, and engineer comeraderie.

More than 120 classes!

- Debugging
- C/C++
- Verification/Validation
- Embedded Internet
- Java

Thousands of products on display

- Microprocessors/Controllers
- Emulators
- Development Tools
- Operating Systems

Call for your free course catalog!

888-239-5563 415-538-3848 outside the U.S. fax: 415-278-5366

email: esc@mfi.com web: www.embedded.com

Send me the details! I am interested in: □ Attending □ Exhibiting □ Receiving conference e-mail updates

EMBEDDED SYSTEMS CONFERENCE

Name

Title Company

Address

City State Zip

Phone Fax E-Mail

Embedded Systems Conference • Miller Freeman, Inc. • 525 Market Street, Suite 500 • San Francisco, CA 94105

SP797

CURTIS/PMC is a global electronics company and a leader in the design and manufacture of control devices for the electric vehicle market.

Software Engineer

Experienced software engineer position for real time coding on microprocessor based motor controllers. BS degree or equivalent plus 4 years of applicable design experience and strong skills in C and Assembly level programming. Experience with feedback control systems and digital hardware a plus.

Senior Software Engineer

Experienced senior software engineer position for real time coding on microprocessor based motor controllers. BS degree or equivalent plus 6 years of applicable design experience in a product development environment required. Strong skills in C and Assembly level programming. Experience with feedback control systems and digital hardware a plus.

We offer company paid health, tuition, and vacation benefits, profit sharing and 401(k) plans. For consideration, qualified candidates should send or fax their resume to: **Human Resources Department, CURTIS/PMC, 6591 Sierra Lane, Dublin, CA 94568. Fax: (510) 833-8777. Email: curtisca@ccnet.com.** Equal opportunity employer m/f/d/v.

CURTIS PMC

A DIVISION OF CURTIS INSTRUMENTS, INC.

Looking for new *Challenges?*

If you have an engineering background and good communication skills, then you may have what it takes to enter the exciting world of magazine publishing.

Embedded Systems Programming seeks a **Technical Editor** for its San Francisco office. The ideal candidate will have strong software and hardware skills, design or applications engineering experience, the ability to translate technical information into fluid prose, and the ability to meet rigorous deadlines. The position entails writing and editing technical articles and product reviews, working with the Editor on technical aspects of magazine and conferences, and meeting with vendors. Prior writing experience is a plus.

If interested, please send resume and cover letter (including salary requirements) to:

JQS/Tech • Miller Freeman Human Relations
525 Market St., Suite 500 • San Francisco, CA 94105
fax: 415.278.5341

Attaining new heights...
Aerospace Controls.

The thrust of tomorrow...
Engine Controls.

Driving the future...
Transportation Controls.

Winning new business is like soaring across the horizon...A path that knows no boundaries. A journey defined only by creativity, At **Lockheed Martin Control Systems**, we are winning new business and looking for talented people to join in our success.

If you seek an exciting career in real-time embedded systems, a rewarding environment, cutting-edge challenges, and state-of-the-art technology, then come to LMCS. We offer unlimited potential in both military and commercial markets. **Climb into the driver's seat!**

- **Software Engineering**
  Ada, JOVIAL, FORTRAN, C/C++
- **Software Quality Assurance Engineering**
- **Systems Engineering**

We offer a competitive compensation package reflective of our leadership status. For consideration, please send/fax your resume to: **Lockheed Martin Control Systems, Human Resources Dept. OA9707a, 600 Main St., Johnson City, NY 13790. FAX: (607) 770-2567. E-mail: cs.staff1@lmco.com** Employment is contingent upon passing a pre-employment drug test provided by Lockheed Martin. We support a drug free/smoke free workplace and are an equal opportunity employer, m/f/d/v. Women and minorities are encouraged to apply.

LOCKHEED MARTIN
Mission Success

PROJECT LEADER EMBEDDED SYSTEMS

Instromedix has an immediate need for a software engineer to lead their team. We are a rapidly growing, medical systems company with 27+ years of experience. The company's state of the art electronic products and systems for monitoring cardiac patients are used by patients, physicians, and hospitals throughout the world.

We seek an experienced software engineer who has demonstrated the ability to lead a multi-disciplined, project team. A BS or MS degree in Computer Science (or equivalent) with 2+ years C and assembler programming experience is required. Medical equipment design experience and exposure to Motorola processors are desired. Experience with OOD and C++ will be considered a plus.

Instromedix is located 10 miles west of Portland, in Hillsboro, OR. The company offers excellent compensation and benefits package and is an Equal Opportunity Employer. All interested candidates should send cover letter and resume to: Instromedix, c/o the MBL Group, Inc. 522 SW Fifth Avenue Suite 1100, Portland, OR 97204, or fax (503) 224-6707, email paul@mblgroup.com

INSTROMEDIX
www.imxcorp.com

EMBEDDED SYSTEMS ENGINEER

Sunair Electronics is seeking an embedded systems engineer responsible for the design, engineering and implementation of microprocessor control systems for high frequency single side-band radio and data terminal equipment.

Successful candidates will have a B.S.E.E. and experience with the following:

- Total project management, including the ability to analyze requirements, develop system specifications and architectural design.
- C Software.
- Embedded hardware design.
- Real Time Operating Systems.
- Serial communications protocols.

Familiarity with Analog Devices DSP hardware, Intel 80C188 and radio communications systems considered a plus.

Sunair Electronics manufactures and markets strategic HF SSB communications equipment and systems for domestic and international military/government clients.

Sunair Electronics incorporated in 1956, is a public company, traded on the American Stock Exchange, symbol SNR.

Interested candidates should forward resumes to:
Sunair Electronics, Inc.
Attention: Eng. Dept.
3101 SW 3 Avenue • Fort Lauderdale, FL 33315
Fax: 954.765.1322 • e-mail: sunair@safari.net

WE ARE WIND RIVER SYSTEMS

We are actively **developing** new products and technology that will be used to build the infrastructure for the 21st century. **We are** the world leader with our RTOS embedded software development and tools. Help define the next generation I2O System.

We are a company with:

Quality
Integrity
High Growth
High Performance Stock
Volleyball/Softball
3 Weeks Vacation
5-10 Minute Access to SF
Intelligence
Thoughtfulness
Bleeding Edge
Recreation Rooms
Hot Tub
5 Month Sabbatical
Opportunity to Grow

The engineers that we sell to are extremely pleased with our Tornado and Vx Works products... (ask them!)

We invite you to review our www.wrs.com site for comprehensive list of all our open positions and everything you ever wanted to know about Wind River Systems. **We are** confident that you will be impressed.
We have 20 openings in engineering (at all levels including a couple of Director positions):

Strategic Integration Group Manager
Compiler Engineer
Windows Engineer
Network Engineer
Engineering Services
Software Engineering Instructor
Technical Support Engineers
Technical Sales
& Field Support Engineers
OS Engineers
Director of Customer Support
Director of Programs

For immediate consideration, please fax/email or snail mail your resume to: (510) 749-2302 Attn: HR, email: Laurie.Harper@wrs.com, or
1010 Atlantic Ave., Alameda, CA 94501.
equal opportunity employer

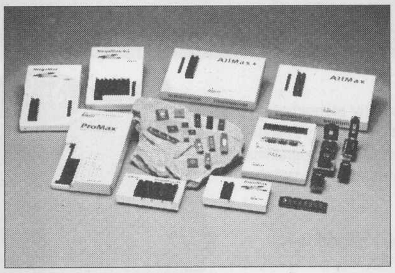
WindRiver
SYSTEMS

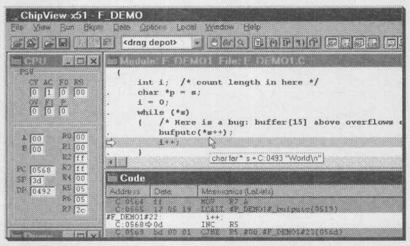
http://www.wrs.com

EMBEDDED MARKETPLACE

To find out more about the products and services displayed here, circle the corresponding Reader Service number on the response card bound into this issue. Each advertiser will send you more information free of charge!

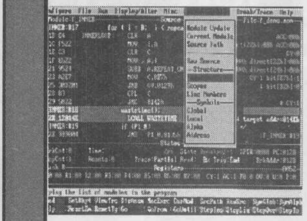
Embedded Marketplace is a special showcase section reserved for advertisers with standard 1/9-page display ads (2 1/4 by 2 inches). For information on placing an ad in Embedded Marketplace, call Robin Lander (415) 278-5274 or Ryan Sorley (617) 235-8255.



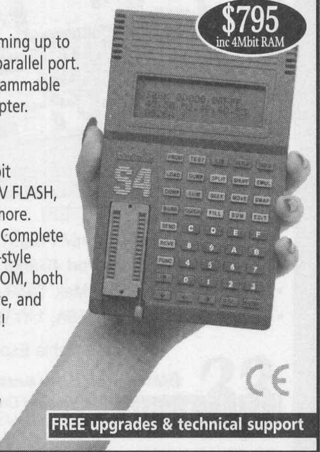

ChipView-x51 - F_DEMO
File View Run Bkpts Data Options Local Window Help
<drag depot>
CPU
PSW
CY AC F0 RS
0 1 0 00
OV F1 P
0 0 0
A 00
B 00
PC 0568
SP 3d
DP 0492
R0 00
R1 00
R2 ff
R3 ff
R4 00
R5 05
R6 05
R7 2c
Module: F_DEMO1 File: F_DEMO1.C
char far * s = C:0493 "World\n"
Code
Address Data Mnemonics (Labels)

```
{
    int i;  /* count length in here */
    char *p = s;
    i = 0;
    while (*s)
    {   /* Here is a bug: buffer[15] above overflows e
        bufputc(*s++);
        i++;
}
```

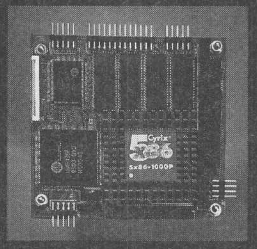
```
C:0564  ff         MOV   R7,A
C:0565  12 05 19   LCALL #F_DEMO1#_bufputc(0519)
#F_DEMO1#22:          i++;
C:0568  0d         INC   R5
C:0569  bd 00 01   CJNE  R5,#00,#F_DEMO1#23(056d)
```



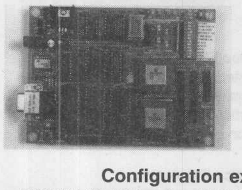
$1249
Dataman
INTELLIGENT UNIVERSAL PROGRAMMER

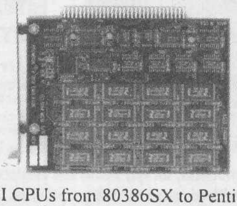


$795
inc 4Mbit RAM
S4

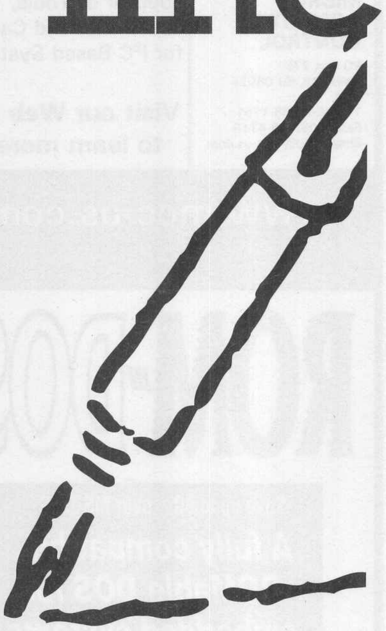


Cyrix
586
5x86-100QP

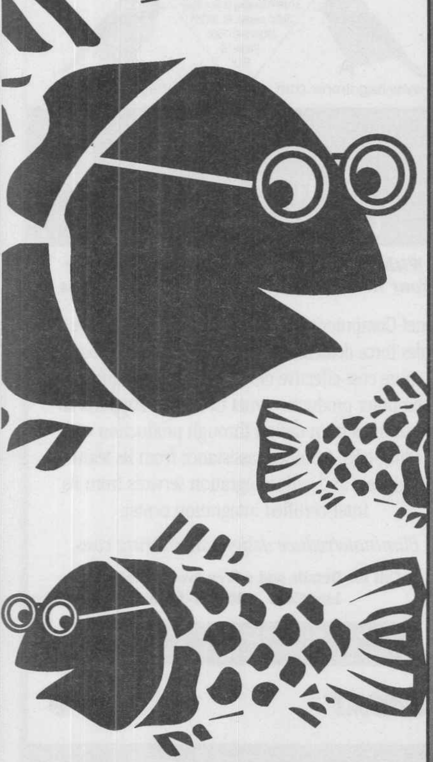




FROM $95.00



Big Fish read



Embedded Systems
PROGRAMMING

Advertise in the *Embedded Systems Programming* Career Section!

Call today!

WEST • Robin Lander
415.278.5274

EAST • Ryan Sorley
617.235.8255

Emulators for 8051/52, 251, 196, PIC and PowerPC

In-Circuit Emulators Support

PowerPC series,	ColdFire series,
251 series,	196 series,
80C186/188 series,	Am186EX series,
68302 series,	68306,
68307,	PIC 16F series,
8051/52	

- **Real-time In-Circuit Debugging and Development**
- **Windows-based Source Level Debugger**
- **Software Performance Analysis**
- **Trace and Trigger on-the-fly**

OEM Deal Welcome

Looking for Local Reps and Distributors.

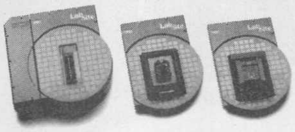
More Information, Please Contact

Microtek San Jose Division
Tel: 408-955-0225
Fax: 408-955-9705
E-mail: mice-sale@micrtek.com
Visit for Demo and Detail
http://www.microtek.com.tw/mice

MICROTEK

CIRCLE #87 ON READER SERVICE CARD

Programming power
plus world class support starting at $1995



The new LabSite™ universal device programmer from Data I/O:

- Support for thousands of popular logic, memory and microcontroller devices
- One year of FREE algorithm updates delivered to you automatically
- Easy-to-use Windows® interface
- Optional socketing systems support PLCCs, QFPs, TSOPs, and other fine-pitch packages
- Worldwide service and technical support

Call for your free demo software
1-800-332-8246
Ext. 850
http://www.data-io.com

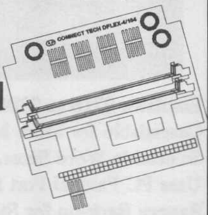
DATA I/O

1-800-3-DATAIO

CIRCLE #88 ON READER SERVICE CARD

NEW

DFlex-4/104
PC/104 Multiport Serial Board



- 4 asynchronous ports
- 16550 UARTs with 16 bytes of FIFO
- RS-232, RS-485/422 RS-423 & 20 mA interfaces
- up to 115K baud on all ports
- user selectable port addresses and IRQs
- PC/104 form factor
- multiple boards can reside in a system for large applications

Call today for more information:

Connect Tech Inc.
727 Speedvale Ave. West
Guelph, Ont. Canada N1K 1E6
Tel: 519.836.1291 Email: sales@connecttech.com
Fax: 519.836.4878 HTTP: //www.connecttech.com

CIRCLE #89 ON READER SERVICE CARD

MULTITASKING

OPERATING SYSTEM

byte BOS™

DOS PC
80X86
8096 8051
68XXX
68HCXX
64180 630X
H85XX
H83XX
37700 C16x
DSP C2x C5x
DSP C3x C4x
ARM THUMB

www.bytebos.com

800-788-7288

CIRCLE #90 ON READER SERVICE CARD

Fast. Reliable. Affordable.

NEEDHAM'S DEVICE PROGRAMMERS are the easiest and most cost-effective way to read, program and verify 2716 – 8 meg EPROMS. Support for Micros, Flash, EPROM, 16-bit, PLDs, and Mach (call for support list for specific models, or download demos from our BBS or web site). Easy to use menu driven software features on-line help, and a full-screen editor. Support for macros, read and save to disk, and split and set options.

- **Free technical support • Free software upgrades**
- **1 to 2 year warranty on all parts and labor**
- **30-day money-back guarantee • Made in the U.S.A.**
- **All models include software, on-line help, cables, and power transformers (where applicable)**

NE **(916) 924-8037**

NEEDHAM ELECTRONICS, INC.
4630 Beloit Drive, #20, Sacramento, CA 95838
FAX (916) 924-8065 • BBS (916) 924-8094
(Mon. – Fri. 8 am – 5 pm, PST) http://www.needhams.com/

CIRCLE #91 ON READER SERVICE CARD

Advertiser Index

ADVERTISER	PAGE	R.S. #
Accelerated Technology	65	31
Adtron Corp.	86	45
Advanced Transdata Corp.	119	106
Advantech	110	109
Advin Systems Inc.	114	71
American Arium	119	104
American Arium	94	54
American Predator Corp.	68	35
Applied Microsystems Corp.	10-11	6
Archimedes Software	98	56
Arcom Control Systems	38	16
Atlas Computer Equipment	119	103
Atmel Corp.	17	*
Avnet Computer	118	97
Avocet Systems Inc.	64	30
Avocet Systems Inc.	118	98
Axiom	52	21
Bagotronix	118	94
Beacon Development Tools	36-37	15
BP Microsystems	C2	1
Byte Bos Integrated Systems	117	90
Byte Craft Ltd.	68	34
CAD-UL	114	73
ChipTools	115	79
CMX Company	88	47
Concurrent Sciences Inc.	51	20
Connect Tech Inc.	117	89
Coreco Inc.	86	111
Cosmic Software	124	63
Curtis PMC	112	*
Cypress Semiconductor	45	*
Data I/O Corp.	117	88
Datalight	115	78
Dataman Programmers Inc.	116	82
DFI	119	100
Diab Data	97	55
EBS	101	59
Electronic Engineering Tools	115	76
EMAC Inc	88	48
Embedded Systems Conference	111	*
Embedded System Products	70	37
Embedtec	118	96
Empress Software	62	28
EPI	92	52
Epilogue Technology Corp.	71	38
EST Corp.	1	2
Express Logic	13	7
General Software	116	83
Go DSP Corp	122	62
Grammar Engine Inc.	90	50
Green Hills Software Inc.	103	60
Hewlett Packard	33	13
Hewlett Packard	79	42
Hitachi America	18	*
HITECH Equipment Corp.	66	32
Hitex	61	27
Huntsville Microsystems Inc.	C4	66
I Logix	69	36
IAR Systems	75	41
ICSPAT Conference	125	*
Instromedix	113	*
Integrated Systems Inc.	8	5
KADAK Products	28	10
Keil Software Inc.	6	4
Kontron Elektronik	39	17
Lauterbach	91	51
Lockheed Martin	112	*
Lynx Real Time Systems	55	24
Metalink Corp.	115	81
Micro Computer Control	115	75
Micro Computer Specialists Inc.	116	86
Micro Digital Inc.	74	40
Micro/Sys Inc.	98	57
Microchip Technology	87	46
Microsystems Development	116	85
Microsystems Development	118	95
Microtec	63	29
Microtek International Inc.	117	87
Microtek International Inc.	119	101
Microtek International Inc.	127	64
Miller Freeman Publications	109	*
MiTAC Industrial Corp.	118	93
Motorola	23	*
Motorola	4-5	*
Needham's Electronics	117	91
Nohau Corp.	114	70
Nohau Corp.	93	53
Nohau Corp.	C3	65
ObjectTime Limited	57	110
Octagon Systems	89	49
On Time	50	*
Pacific Softworks	99	58
Paradigm Systems	72	*
Parallax	118	92
Pentica Systems	73	39
Philips Semiconductor	21	9
Precise Software	60	26
Premia	105	61
QNX	47	18
QNX	83	43
Radisys	35	14
Raima Corp.	41	*
R&D Books	109	*
Realtime Control Inc.	118	108
Scanlon Design Inc.	115	80
Siemens	14-15	8
Signum Systems	114	68
Softaid	106	112
Softchip Israel Ltd.	56	25
Softools Inc.	114	69
Software Development Systems	2	3
Spectrum Digital	29	11
SSV Software Systems	116	84
Sun Air Electronics	113	*
TASKING	49	19
TASKING	84	44
TechTools	115	77
TechTools	118	99
Tern Inc.	119	102
Tribal Microsystems	114	67
U.S. Software Corp.	67	33
V Automation	54	23
Vesta Technology	119	107
White Mountain DSP	31	12
Wind River Systems	113	*
Wind River Systems	53	22
Xeltek	114	74
Z World Engineering	114	72

The index on this page is provided as a service to readers. The publisher does not assume any liability for errors or omissions.

* These advertisers prefer to be contacted directly.

STATE OF THE ART

by P.J. Plauger

Floating-Point Primitives

The C library has always included seminumerical functions for manipulating floating-point values. These functions have names such as `ldexp`, `frexp`, and `modf`, all declared in the header `<math.h>`. You may never have called them directly, but you are likely to have profited from their presence when you computed a cosine or an exponential. Early math functions were more robust and portable for having been written in terms of these primitives.

Embedded systems profit from this approach as well. Avoiding actual floating-point operations is often simpler and faster, particularly on microprocessors with little or no hardware support for floating-point arithmetic. Even if you have such hardware, using seminumerical functions eases handling of special cases, such as overflow and underflow. Just trapping out is not always convenient, and simply ignoring a bogus result is seldom wise.

IEEE 754 floating-point arithmetic adds even more complexities to the game. It demands a slightly different set of primitives for dealing with infinities, non-a-number codes, and other subtleties. This column describes a set of primitives that I have found convenient for writing the math functions of the Standard C library. They were designed with IEEE 754 in mind, but they are not limited to that encoding. I have used these primitives to write all the math functions presented in my book *The Standard C Library* (Prentice Hall, 1992). The functions are demonstrably portable and arguably efficient and easy to read.

Early math functions were more robust and portable for having been written in terms of these primitives.

I have found the primitives to be useful in knocking together other math functions as well. I present them here in the hopes that you might also find them useful. I find math functions aren't nearly as intimidating to write if you don't have to depend on existing floating-point hardware or software emulators to handle all the corner cases the way you need. Floating-point arithmetic can generate a number of exceptional conditions:

- Overflow occurs when the magnitude of the result is too large to represent
- Underflow occurs when the magnitude of the result is too small to represent
- Loss of significance occurs when the magnitude of a sum or difference is much smaller than that of either operand
- A domain error (such as zero divide) occurs when you specify a combination of operands for which the operation is not defined

Even the most naive programmers soon learn to worry about these exceptions. When they occur, different computers do different things. Some terminate program execution abruptly. Others continue, but make some attempt to signal the error. These exceptions may set an indicator (such as the notorious `errno` of the Standard C library, declared in `<errno.h>`). They may produce a result that:

- Unequivocably signals a problem (such as some special code)
- Equivocably signals a problem (such as zero or `HUGE_VAL`, declared in `<math.h>`)
- Is pure garbage (such as an oversize exponent that wraps around)

If your goal is to write code that is both robust and portable, this spectrum of possibilities is dismaying. You soon learn that the only safe way to deal with exceptions is to avoid them. Test before you compute. Make sure that all floating-point operations produce unexceptional results. At the very least, localize the points at which exceptions can occur. Doing so lets you isolate any special code you must introduce to handle the exceptions uniformly across diverse implementations. Localizing lowers the cost of porting code. A well chosen set of primitives can supply just the points you need to handle floating-point exceptions.

One of the most demanding uses for floating-point code is in writing the functions that constitute the math library. In C, these are primarily the functions declared in `<math.h>`. You might also include the functions that convert between text representation and floating-point values. In the Standard C library, these are declared in `<stdlib.h>`. These functions must generate the most precise answers possible for all sensible inputs. They must avoid intermediate exceptions even for the most extreme argument values. And they often must be portable in the bargain, in order to recoup the investment of effort across as many markets as possible.

The C library has striven to meet these goals from the outset. Dennis Ritchie and his friends knew to avoid many of the common pitfalls in writing math functions. While C has had its notorious lapses in this area, it has probably fared better than most programming languages.

Evidence of this heightened awareness lies in the C library itself, in the handful of seminumerical functions I mentioned earlier. These functions are clearly aimed at easing the burden of the cautious numerical programmer. They let you dismantle floating-point values in various ways. You can then work with pieces that are integers or floating-point values with a more restricted range. Finally, you put the pieces back together to develop the ultimate result.

I call these functions "semi-numerical," because they need not execute floating-point instructions to get the job done. You can write them in C or assembly language as if they are operating on arrays of integers. You shift, mask, and merge to manipulate the components of a floating-point value separately. True, some floating-point processors have instructions that do part or all of the job. Even for these machines, however, you might still have occasion to avoid floating-point instructions, as I mentioned earlier. The three most important functions in the semi-numerical group are:

- `double frexp(double x, int *pex)`, which extracts the power-of-two exponent of `x` and stores it in `*pex`, then returns the residual fraction whose magnitude now lies in the half-open interval [$^1/_2$, 1) (or is zero)
- `double ldexp(double x, int ex)`, which multiplies `x` by two raised to the power `ex` and returns the result (hence undoing the damage caused by `frexp`)
- `double modf(doublex, double *pin)`—which extracts the integer part of `x` and stores it in `*pin`, then returns the residual fraction whose magnitude now lies in the half-open interval [0, 1) and whose sign is the same as `*pin` (and `x`)

These operations play a pivotal role in implementing nearly all of the standard math functions declared in <math.h>. Unfortunately, these particular functions don't quite do the whole job. I have written several math libraries over the years, most of them in C. In each case, I ended up writing a different set of math primitives. The three functions shown above always turned out to be easily expressed in terms of the primitives I chose. But I could never quite do the job the other way around.

EXCEPTIONS

The problems lie primarily in the area of exception handling. Any of the three functions can be handed an exceptional argument, at least in principle. The function `ldexp` can generate an overflow or underflow. If the functions don't land on their feet when an exception occurs, you must ensure that they never see exceptions. (Remember what I said earlier about the best way to keep floating-point code portable.)

Because they are so widely used, these are the very functions you want to have to help you write robust code. You don't want to have to call one function to test for exceptions, then another to unpack an operand appropriately. You don't want to have to test first whether repacking is safe, then call a function to do the actual repacking. That's not a good recipe for writing code that is both portable *and* efficient.

Consider the function `ldexp` as the simplest example. This function is often the agent that repacks the components you have manipulated separately (and safely). You can thus make it the one and only place where overflow or underflow can occur. As a seminumerical function, `ldexp` can detect an impending exception without tickling the dragon's tail. You can steer well clear of any hardware traps when you write the function.

Unfortunately, you have only limited latitude in how you write `ldexp`. The C Standard dictates its outward behavior. The function can (and must) set `errno` on a range error; it can (and must) substitute `HUGE_VAL` or zero for an unrepresentable result when a range error occurs. But the function has no

nice way to tell the caller that it has done so. Comparing the return value against `HUGE_VAL` and zero can be both time consuming and inconclusive.

Now consider what `frexp` should do when handed the value `HUGE_VAL`. On some machines, this is just a very large representable value. It *can* be represented as a power-of-two exponent and a fraction. But should it? The C Standard doesn't really say. You probably want an unpacking primitive that is smarter, and more informative, than `frexp` is allowed to be. That's the tip of the iceberg. The real danger to shipping lies in the complexities introduced with the IEEE 754 Standard for floating-point arithmetic. That standard introduces all sorts of codes for exceptions. Besides being a finite, representable value, a floating-point operand can be:

- a *signalling NaN* ("not-a-number") that should raise an immediate exception for any operation except a simple copy
- a *quiet NaN* that should percolate through to the result wherever possible
- Inf (for "infinity"), either positive or negative
- zero, either positive or negative

The C Standard suffered a few last-minute edits designed to make it tolerant of IEEE 754 arithmetic. `HUGE_VAL` can be represented as Inf, for example. Domain errors can be represented in a variety of ways, probably including a NaN result. Those changes are necessary, but they are not sufficient. An implementation of C that endeavors to support IEEE 754 arithmetic has little guidance from the C Standard.

More recently, proposed extensions to the revised C Standard have addressed some of these issues. For IEEE 754 arithmetic, at least, there is now more guidance as to how to compare floating-point values in the presence of NaNs. Inf and zero, of either sign, have sensible orderings defined for the comparison operators. NaNs do not. Hence, the expression $x<y$ is properly neither true nor false if either operand is a NaN. It would be better, in many ways, for the program to raise an immediate exception to handle a NaN when executing such an expression.

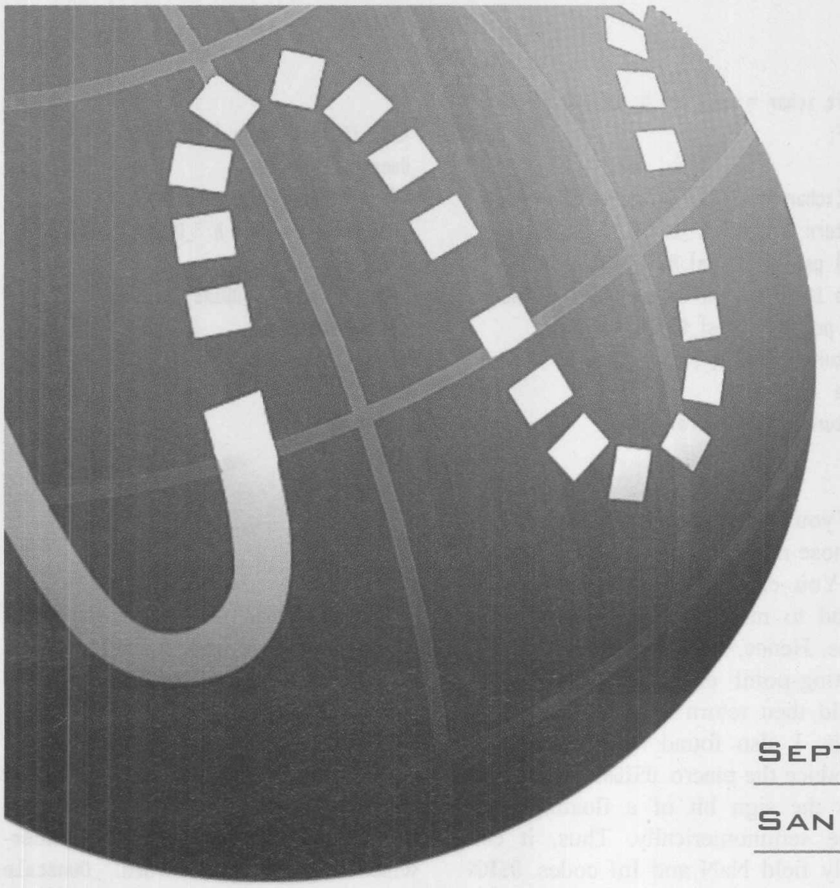
A common convention is to have the expression $x==x$ be false if x is a NaN. Such notation makes me queasy, particularly buried inside a complex algorithm. It looks too much like a tautology gone astray. Other people have proposed introducing a slew of additional comparison operators to the C language. In one scheme, an operator that begins with a bang *!* tolerates NaNs. Thus, $x \,!\!< y$ is true if y is greater than or equal to x or if either operand is a NaN. Fortunately, this proposal was rejected by WG14, the committee revising the existing C Standard.

Using operator notation this way eliminates the need to raise exceptions, but at the cost of complexifying C even further. People might even start accusing C expressions of being cryptic. In my experience, such an approach is neither necessary nor sufficient, because:

- You need to do more with NaNs than simply copy them or compare them safely. You may need to distinguish quiet and signaling NaNs, for example. You always want to treat them quite differently from other operands
- You often need to treat Inf quite differently from other operands
- You may want to distinguish plus zero from minus zero in some contexts (although personally I have reservations about the utility of minus zero)

I have found a different approach more useful. I find frequent occasion to categorize a floating-point value before I muck with it. At the very least, I want to distinguish between:

- `NAN` for a quiet NaN
- `INF` for Inf of either sign
- zero for zero of either sign
- `FINITE` for a finite, representable value of either sign



BETTER PRIMITIVES

Here is the simplest possible example. The function `fabs` looks to be trivial. In principle, one could write it as:

```
double fabs(double x)
{ /* compute absolute value */
return (x < 0.0 ? -x : x);
}
```

In practice, this code is a sucker for NaNs. I introduced the function `_Dtest(double *px)` which categorizes `px` seminumerically. The header `"xmath.h"` declares `_Dtest` and defines macros for the integer category codes it returns, as indicated above. (In principle, the return value is an enumeration.) Now `fabs` can be written safely as:

```
#include "xmath.h"
double fabs(double x)
 { /* compute absolute value */
 switch (_Dtest(&x))
  { /* test for special codes */
 case NAN:
  errno = EDOM;
  return (x);
 case INF:
  errno = ERANGE;
  return (_Inf._D);
 case 0:
  return (x);
 default: /* finite */
  return (x < 0.0 ? -x : x);
  }
 }
```

It's not nearly as fast or elegant as the obvious version, but it works better. Here is what `_Dtest` looks like. The various funny macro names that begin with `_D` define machine-dependent properties of the floating-point representation. They correct for changes in byte order among various IEEE 754 implementations. They also tolerate a few similar formats, such as the PDP-11/VAX-11 floating-point format:

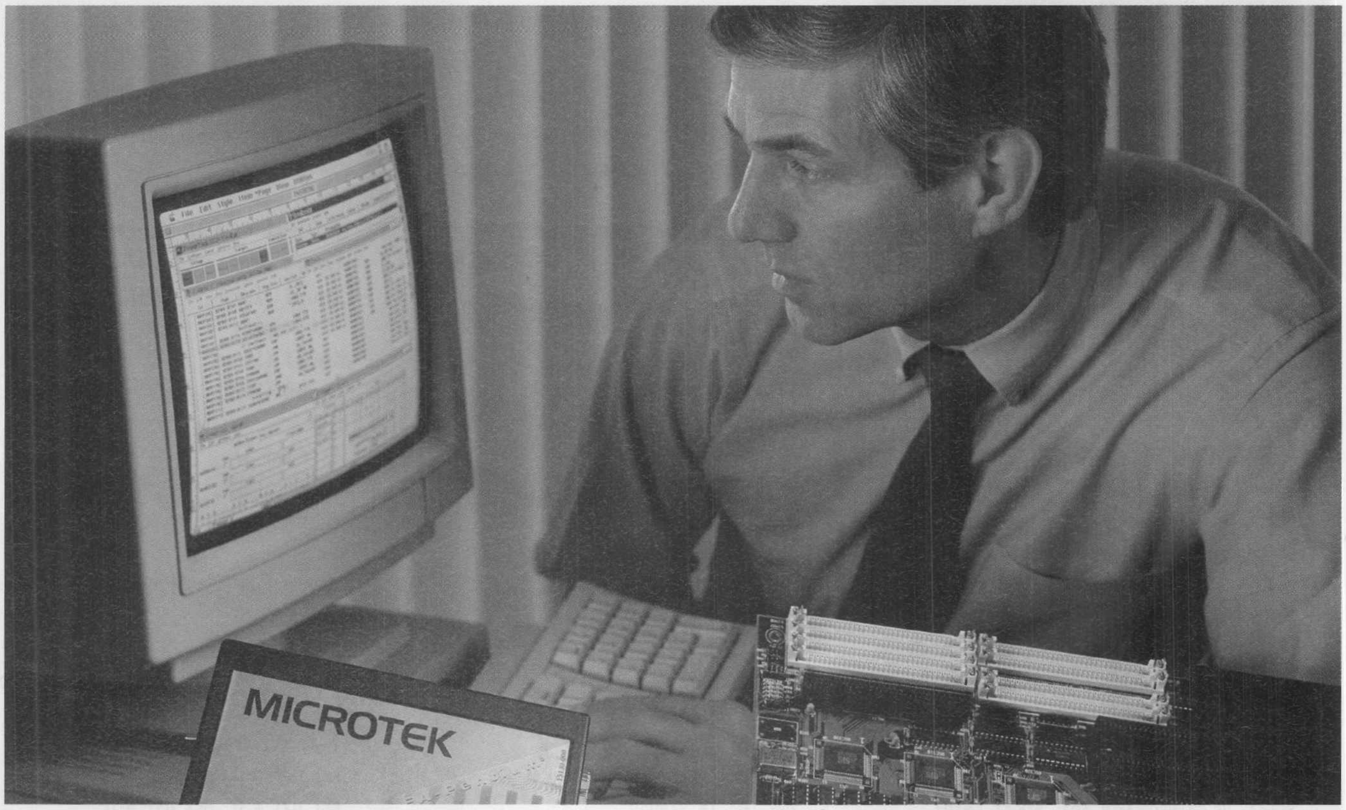
```
/* _Dtest function -- IEEE 754 version */
#include "xmath.h"
short _Dtest(double *px)
 { /* categorize *px */
 unsigned short *ps = (unsigned short
*)px;
 short xchar = (ps[_D0] & _DMASK) >>
_DOFF;

 if (xchar == _DMAX) /* NaN or INF */
  return (ps[_D0] & _DFRAC || ps[_D1]
   || ps[_D2] || ps[_D3] ? NAN : INF);
 else if (0 < xchar || ps[_D0] & _DFRAC
  || ps[_D1] || ps[_D2] || ps[_D3])
  return (FINITE); /* finite */
 else
  return (0); /* zero */
 }
```

If you want to handle signaling NaNs (I chose not to), here is the place to do so. You call `_Dtest` only when you intend to muck with a floating-point value. Hence, this function can raise a floating-point exception for you. It would then return `NAN` only for quiet NaNs. I also found it convenient to introduce the macro `DSIGN`. This macro tests the sign bit of a floating-point value seminumerically. Thus, it can safely field NaN and Inf codes. `DSIGN` can also correctly distinguish plus and minus zero, a distinction otherwise difficult to make with C comparison operators. The other math primitives follow the model established by `_Dtest`. All are tolerant of the various IEEE 754 exception codes. All attempt to do something sensible with these various codes. All return a category code to guide the caller in its subsequent actions. Thus, it is necessary to call `_Dtest` only when none of the other common primitives are needed. Here, for example, is a more robust substitute for `frexp`. The function `_Dunscale` unpacks an operand only if it is finite. Otherwise, it returns the appropriate category code:

```
#include "xmath.h"
short _Dunscale(short *pex, double *px)
 { /* separate *px to 1/2 <= |frac| < 1
and 2^*pex */
 unsigned short *ps = (unsigned short
*)px;
 short xchar = (ps[_D0] & _DMASK) >>
_DOFF;

 if (xchar == _DMAX)
  { /* NaN or INF */
  *pex = 0;
  return (ps[_D0] & _DFRAC || ps[_D1]
   || ps[_D2] || ps[_D3] ? NAN : INF);
  }
 else if (0 < xchar || (xchar =
_Dnorm(ps)) != 0)
  { /* finite, reduce to [1/2, 1) */
  ps[_D0] = ps[_D0] & ~_DMASK | _DBIAS <<
_DOFF;
  *pex = xchar - _DBIAS;
  return (FINITE);
  }
 else
  { /* zero */
  *pex = 0;
  return (0);
  }
 }
```

This function must also deal with another added complexity of IEEE 754 arithmetic. A value with very small magnitude can be "denormalized." That provides a form of "gradual underflow" that has desirable properties in a few cases. It also mucks up some functions that would be otherwise fairly straightforward. `_Dunscale` calls the function `_Dnorm` to deal with denormalized values. The latter function produces a normalized fraction, if possible. It also returns a correction to the power-of-two exponent for a finite denormalized operand:

```
#include "xmath.h"
short _Dnorm(unsigned short *ps)
 { /* normalize double fraction */
 short xchar;
 unsigned short sign = ps[_D0] & _DSIGN;

 xchar = 0;
 if ((ps[_D0] &= _DFRAC) != 0 || ps[_D1]
  || ps[_D2] || ps[_D3])
  { /* nonzero, scale */
  for (; ps[_D0] == 0; xchar -= 16)
   { /* shift left by 16 */
   ps[_D0] = ps[_D1], ps[_D1] = ps[_D2];
   ps[_D2] = ps[_D3], ps[_D3] = 0;
   }
  for (; ps[_D0] < 1<<_DOFF; --xchar)
   { /* shift left by 1 */
   ps[_D0] = ps[_D0] << 1 | ps[_D1] >> 15;
   ps[_D1] = ps[_D1] << 1 | ps[_D2] >> 15;
   ps[_D2] = ps[_D2] << 1 | ps[_D3] >> 15;
   ps[_D3] <<= 1;
   }
  for (; 1<<_DOFF+1 <= ps[_D0]; ++xchar)
   { /* shift right by 1 */
   ps[_D3] = ps[_D3] >> 1 | ps[_D2] << 15;
```

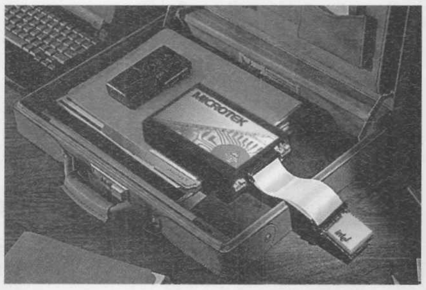


"If I could find it, I could fix it."

SEE your code at work. Find timing conflicts fast that are invisible with logic analyzers and software debuggers.

Microtek In-Circuit Emulators combine a state-of-the-art source level debugger with the most advanced event trigger and trace system available.

160-bits Wide by 256k Trace with Clock-edge Resolution. With this much trace and smart triggering, you can record virtually every event. And events can be followed right back to their source code without stopping the target.



Smaller is Better. Compare today's Microtek In-Circuit Emulators that fit in your briefcase, with the traditional "chassis" of just two years ago. The difference is remarkable! The PowerPack®EA for the Pentium® processor is only 7.2" x 4.6". And the probe tip is smaller than your business card, so it fits into the tightest targets.

Call for FREE ICE Tips AppNotes:

1 (800) 886-7333

www.microtekintl.com
Phone: (503) 645-7333
Fax: (503) 629-8460

SWAT™Software Analysis Tool

- Fast, easy code validation for design engineers.
- Built into Microtek In-Circuit Emulators.
- Code Coverage and Performance Analysis offer you design spontaneity without instrumenting your code!

CODE CHANGES

MICROTEK
IN-CIRCUIT EMULATORS

NEW PRODUCTS
Three Emulators for Pentium® Processors
SWAT™ Software Analysis Tool
Two National NS486™ Emulators
High-Performance 80C186 Emulator

Microtek In-Circuit Emulators for the following processors:

Pentium®• Intel486™• National NS486™• Intel386™EX • 386DX • 386CX/SX • 80C186 • 8051
68360 • 68340 • 68F333 • 68332 • 68331 • 68330 • 68HC16 • 68328 • ColdFire

CIRCLE # 64 ON READER SERVICE CARD

Pentium, Intel486 and Intel386 are registered trademarks of Intel Corp. NS486 is a registered trademark of National Semiconductor. PowerPack is a registered trademark of Microtek International. © 1997 Microtek International

```
    ps[_D1] = ps[_D1] >> 1 | ps[_D0] << 15;
    ps[_D0] >>= 1;
    }
   ps[_D0] &= _DFRAC;
   }
  ps[_D0] |= sign;
  return (xchar);
  }
```

REMAINING PRIMITIVES

The analog of `ldexp` is even messier. `_Dscale` must test for all the usual exception codes in its argument `*px`. It must also generate infinities and denormalized values. The code that follows is safe against intermediate integer overflow so long as `short` has a smaller representation than `long`:

```
#include "xmath.h"
short _Dscale(double *px, short xexp)
 { /* scale *px by 2^xexp with checking */
 long lexp;
 unsigned short *ps = (unsigned short
*)px;
 short xchar = (ps[_D0] & _DMASK) >>
_DOFF;

 if (xchar == _DMAX) /* NaN or INF */
  return (ps[_D0] & _DFRAC || ps[_D1]
   || ps[_D2] || ps[_D3] ? NAN : INF);
 else if (0 < xchar)
  ; /* finite */
 else if ((xchar = _Dnorm(ps)) == 0)
  return (0); /* zero */
 lexp = (long)xexp + xchar;
 if (_DMAX <= lexp)
  { /* overflow, return +/-INF */
  *px = ps[_D0] & _DSIGN ? -_Inf._D :
_Inf._D;
  return (INF);
  }
 else if (0 < lexp)
  { /* finite result, repack */
  ps[_D0] = ps[_D0] & ~_DMASK |
(short)lexp << _DOFF;
  return (FINITE);
  }
 else
  { /* denormalized, scale */
  unsigned short sign = ps[_D0] & _DSIGN;

  ps[_D0] = 1 << _DOFF | ps[_D0] & _DFRAC;
  if (lexp < -(48+_DOFF+1))
   xexp = -1; /* certain underflow */
  else
   for (xexp = lexp; xexp <= -16; xexp +=
16)
    { /* scale by words */
    ps[_D3] = ps[_D2], ps[_D2] = ps[_D1];
    ps[_D1] = ps[_D0], ps[_D0] = 0;
    }
   if ((xexp = -xexp) != 0)
    { /* scale by bits */
    ps[_D3] = ps[_D3] >> xexp
     | ps[_D2] << 16 - xexp;
    ps[_D2] = ps[_D2] >> xexp
     | ps[_D1] << 16 - xexp;
    ps[_D1] = ps[_D1] >> xexp
     | ps[_D0] << 16 - xexp;
    ps[_D0] >>= xexp;
    }
   }
  if (0 <= xexp && (ps[_D0] || ps[_D1]
   || ps[_D2] || ps[_D3]))
   { /* denormalized */
   ps[_D0] |= sign;
   return (FINITE);
   }
  else
   { /* underflow, return +/-0 */
   ps[_D0] = sign, ps[_D1] = 0;
   ps[_D2] = 0, ps[_D3] = 0;
   return (0);
   }
  }
 }
```

The final primitive is the analog of `modf`. I found it useful to make `_Dint` somewhat more general. Some math functions preserve one or more fraction bits while dropping the rest. Thus, negative values of the argument `xexp` specify how many bits to keep to the right of the binary point. Less significant fraction bits are cleared. Note that this function returns the proper category code for the fraction that is discarded, not the integer that is retained. As dizzying as that may appear to be, it proves to be the best behavior for the function:

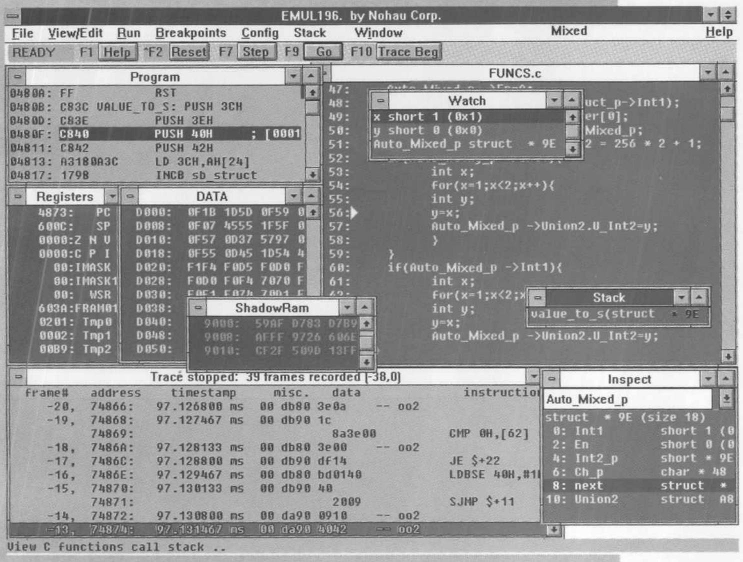
```
#include "xmath.h"
short _Dint(double *px, short xexp)
 { / *test and drop (scaled) fraction bits
*/
 unsigned short *ps = (unsigned short
*)px;
 unsigned short frac = ps[_D0] & _DFRAC
  || ps[_D1] || ps[_D2] || ps[_D3];
 short xchar = (ps[_D0] & _DMASK) >>
_DOFF;

 if (xchar == 0 && !frac)
  return (0); /* zero */
 else if (xchar != _DMAX)
  ; /* finite */
 else if (!frac)
  return (INF);
 else
  { /* NaN */
  errno = EDOM;
  return (NAN);
  }
 xchar = (_DBIAS+48+_DOFF+1) - xchar -
xexp;
 if (xchar <= 0)
  return (0); /* no frac bits to drop */
 else if ((48+_DOFF) < xchar)
  { /* all frac bits */
  ps[_D0] = 0, ps[_D1] = 0;
  ps[_D2] = 0, ps[_D3] = 0;
  return (FINITE);
  }
 else
  { /* strip out frac bits */
  static const unsigned short mask[] = {
   0x0000, 0x0001, 0x0003, 0x0007,
   0x000f, 0x001f, 0x003f, 0x007f,
   0x00ff, 0x01ff, 0x03ff, 0x07ff,
   0x0fff, 0x1fff, 0x3fff, 0x7fff};
  static const size_t sub[] = {_D3, _D2,
_D1, _D0};

  frac = mask[xchar & 0xf];
  xchar >>= 4;
  frac &= ps[sub[xchar]];
  ps[sub[xchar]] ^= frac;
  switch (xchar)
   { /* cascade through! */
  case 3:
   frac |= ps[_D1], ps[_D1] = 0;
  case 2:
   frac |= ps[_D2], ps[_D2] = 0;
  case 1:
   frac |= ps[_D3], ps[_D3] = 0;
   }
  return (frac ? FINITE : 0);
  }
 }
```

That's the lot. For some real-world examples of how these primitives can be used, see my book. ESP

P.J. Plauger is the author of The Standard C Library *(Prentice Hall, 1992).*

nohau

80C196 In-Circuit Emulators



EMUL196. by Nohau Corp.
File View/Edit Run Breakpoints Config Stack Window Mixed Help
READY F1 Help ^F2 Reset F7 Step F9 Go F10 Trace Beg
Program
Registers
DATA
ShadowRam
FUNCS.c
Watch
Stack
Inspect

```
0480A: FF                RST
0480B: C83C VALUE_TO_S: PUSH 3CH
0480D: C83E             PUSH 3EH
0480F: C840             PUSH 40H      ; [0001
04811: C842             PUSH 42H
04813: A3180A3C         LD 3CH,AH[24]
04817: 1798             INCB sb_struct
```

```
  4873:     PC
  600C:     SP
  0000:Z N V
  0000:C P I
    00:IMASK
    00:IMASK1
    00:  WSR
  603A:FRAM01
  0201: Tmp0
  0002: Tmp1
  00B9: Tmp2
```

```
D000:  0F1B 1D5D 0F59 0
D008:  0F07 4555 1F5F 0
D010:  0F57 0D37 5797 0
D018:  0F55 0D45 1D54 4
D020:  F1F4 F0D5 F0D0 F
D028:  F0D0 F0F4 7070 F
D030:
D038:
D040:
D048:
D050:
```

```
9000:  59AF D783 D7B9
9008:  AFFF 9726 606E
9010:  CF2F 509D 13FF
```

```
47:
48:                                 uct_p->Int1);
49:                                 er[0];
50:                                 Mixed_p;
51:                                 2 = 256 * 2 + 1;
52:
53:            int x;
54:            for(x=1;x<2;x++){
55:            int y;
56:            y=x;
57:            Auto_Mixed_p ->Union2.U_Int2=y;
58:            }
59:      }
60:      if(Auto_Mixed_p ->Int1){
61:            int x;
62:            for(x=1;x<2;x
63:            int y;
64:            y=x;
65:            Auto_Mixed_p ->Union2.U_Int2=y;
```

```
x short 1 (0x1)
y short 0 (0x0)
Auto_Mixed_p struct  * 9E
```

```
value_to_s(struct  * 9E
```

```
Trace stopped:  39 frames recorded [-38,0]
 frame#   address    timestamp     misc.  data                instructio
   -20,   74866:   97.126800 ms   00 db80 3e0a     -- oo2
   -19,   74868:   97.127467 ms   00 db90 1c
          74869:                          8a3e00            CMP 0H,[62]
   -18,   7486A:   97.128133 ms   00 db80 3e00     -- oo2
   -17,   7486C:   97.128800 ms   00 db90 df14              JE $+22
   -16,   7486E:   97.129467 ms   00 db80 bd0140            LDBSE 40H,#1
   -15,   74870:   97.130133 ms   00 db90 40
          74871:                          2009              SJMP $+11
   -14,   74872:   97.130800 ms   00 da90 0910     -- oo2
   -13,   74874:   97.131467 ms   00 da90 4042     -- oo2
View C functions call stack ..
```

```
Auto_Mixed_p
struct  * 9E (size 18)
 0: Int1          short 1 (0
 2: En            short 0 (0
 4: Int2_p        short * 9E
 6: Ch_p          char * 48
 8: next          struct  *
10: Union2        struct  A8
```

Real-Time Microprocessor Development Systems

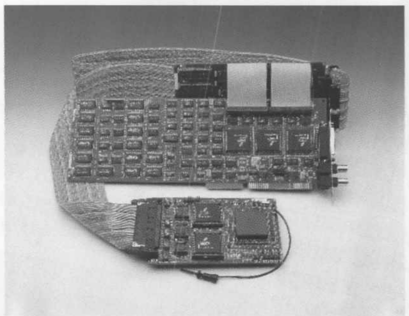
EMUL196™-PC

EMUL196-PC FEATURES:

- Support for 80C196: CA/B, EA, JQ/R/T, KB/C/D, KQ/R/S/T, NP/T/U and many more.
- Real-time emulation at maximum chip speeds.
- Uses bond-out chips for accurate emulation.
- Hosted on PC's and workstations.
- High Level support for popular C-compilers.
- Unlimited hardware breakpoints.
- Break in real-time on Internal Access, both on data value and address.
- Trace board up to 512K deep, 104 bits wide, with 40 bit timestamp. Triggering and filtering with full instruction queue decoding.
- Memory contents shown during real-time emulation (Shadow RAM).
- Code Coverage and Program Performance Analysis.
- CCB's controlled from user interface.
- PCMCIA compliant card for use with laptops.

™ MICROSOFT WINDOWS/'95 COMPATIBLE

*To learn more, call (408) 866-1820 for a product brochure and a FREE Demo Disk. The Demo can also be downloaded from our web site- **http://www.nohau.com/nohau**.*



For more information via your Fax, call our 24-hour Fax Center at (408) 378-2912.

nohau
CORPORATION

51 E. Campbell Avenue
Campbell, CA 95008-2053
Fax. (408) 378-7869
Tel. (408) 866-1820

Argentina 54 1 312-1079/9103, Australia (02) 654 1873, Austria 0222 27720-0, Benelux (078) 681 61 33
Brazil (011) 453-5588, Canada 514 689-5889, Czech Republic 0202-811536, Denmark 45 43 44 60 10,
Finland 90 777 571, France (1) 69 41 28 01, Germany 07043/40247, Great Britain 01962-733 140,
Greece +30-1-924 20 72, India 0212-412164, Israel 03-6491202, Italy 02 498 2051, Japan (03) 3405-0511,

NOHAU INTERNATIONAL

Korea (02) 784-7841, New Zealand 09-3092464, Norway (+47) 22 67 40 20, Portugal 01 4213141,
Romania 056 200057, Singapore +65 749-0870, S.Africa (021) 234 943, Spain (93) 291 76 33,
Sweden 040-92 24 25, Switzerland 01-745 18 18, Taiwan 02 7640215, Thailand (02) 668-5080.

Support also available for:

8051 P51XA™ MCS®251 68HC11 68HC16 683xx

CIRCLE # 65 ON READER SERVICE CARD

HMI. Putting the power behind PowerPC development.

HMI's SPS-2000 Series PowerPC development tools take emulation technology to a whole new level. True in-circuit emulators, they change easily and economically to support most PowerPC variants from Motorola and IBM via a simple, low-cost pod change. And they have the following standard performance features that make them the industry's most powerful microprocessor development system: • Hardware-based non-statistical software performance analyzer. • Ethernet interface for high-speed connectivity. • High-speed overlay memory for code download, mappable via chip selects. • Shadow RAM for real-time data variable/memory monitoring. • 128K deep trace buffer configurable in up to 8192 separate buffers for windowing events of interest. • 8-level trigger and breakpoint sequencing logic. • Time-based delays for break and trigger points. • Direct variable editing in watch windows. • Native GUI support from multiple host platforms (Windows 3.1x/95/NT, Sun, HP) using SourceGate II, HMI's powerful source-level debugger that is a common user interface for all HMI products. • Multiple object file format support. Use your compiler of choice! • Unlimited CodeView windows allow breakpoints to be set across multiple modules displayed in source, assembly, or a combination of the two. • Can be operated stand-alone with no target system required or can be put in-circuit in place of the processor. • **Free** lifetime technical support (no costly yearly support contracts to worry about—ever!)

HMI ALSO PROVIDES SUPPORT FOR THESE PROCESSORS.

MPC8xx	68302 Family	8051
IBM40x	68306	8085
MPC505	68307	8096 Family
ColdFire	68330	NSC800
	68331	Z80
68000	68332	64180/Z180
68020	68333	6809
68030	68340	68HC11 Family
68040	68349	68HC16
68060	68360	68356

Interested? Call or write and we'll be happy to send you more detail ed information. Can't wait? Visit our Web Site at http//www.hmi.com/ for instant access to our SPS-2000 data sheet and information on our $\$199^{00}$ Background Mode Debugger!

HMI™

HUNTSVILLE MICROSYSTEMS, INC.

P.O. Box 12415
Huntsville, AL 35815
Tel: (205) 881-6005
Fax: (205) 882-6701
sales@hmi.com
http://www.hmi.com/

MOTOROLA
EMBEDDED
PLATINUM DEVELOPER

CIRCLE # 66 ON READER SERVICE CARD